



Developer's Guide Supplement
DBCS, 3270/5250, and Dialogs

EASEL Version 1.1
for OS/2
Extended Edition



Developer's Guide Supplement
DBCS, 3270/5250, and Dialogs

EASEL Version 1.1
for OS/2
Extended Edition

First Edition (October 1990)

This publication could include technical inaccuracies or typographical errors. Changes are made periodically to the information herein; these changes will be incorporated in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Requests for copies of this product and for technical information about the system should be made to your local IBM representative.

© Copyright International Business Machines Corporation 1990. All rights reserved.

© Copyright Easel Corporation 1990.

Note to US Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

IBM and OS/2 are registered trademarks of International Business Machines Corporation in the United States and other countries. EASEL is a registered trademark of Easel Corporation in the United States and other countries.

Preface

This publication is a supplement to the *IBM EASEL® Version 1.1 for OS/2® Extended Edition Developer's Guide*. It describes the IBM EASEL for OS/2 Extended Edition (referred to as EASEL OS/2 EE) support for double-byte character set (DBCS) languages, 3270/5250 communications via a dynamic link library (DLL), automatic creation and deletion of dialogs, and other miscellaneous enhancements.

EASEL OS/2 EE is developed by Easel Corporation, Woburn, MA.

IBM and OS/2 are registered trademarks of International Business Machines Corporation in the United States and other countries.

EASEL is a registered trademark of Easel Corporation in the United States and other countries.

Contents

Chapter 1. DBCS Support	1-1
1.1 Overview	1-2
1.2 Installation	1-3
1.2.1 Machine Requirements	1-3
1.2.2 Program Requirements	1-3
1.2.3 Device Support Limitations	1-4
1.2.4 Installing EASEL OS/2 EE DBCS Support	1-4
1.2.5 Files added for DBCS Support	1-5
1.3 Language Elements	1-6
1.3.1 Code Page Support	1-6
1.3.2 DBCS Characters and Mixed Strings	1-7
1.3.3 Line continuation	1-7
1.3.4 Escape Sequences in String Literals.	1-8
1.3.5 Katakana Mnemonics	1-8
1.4 extract from Action Statement	1-10
1.5 Fonts	1-12
1.5.1 Installation of Resolution Dependent Fonts	1-12
1.5.2 Default Font	1-12
1.5.3 OS/2 Presentation Manager Fonts	1-12
1.5.4 EASEL OS/2 EE Fonts	1-14
1.6 Keyboard Status Display Area	1-15
1.7 Entry Field Data Types	1-16
1.8 Textual Regions	1-17
1.8.1 Moving the Text Cursor and Specifying Column Position	1-17
1.8.2 Block Operations	1-18
1.8.3 Segment Operations	1-19
1.8.4 String Operations	1-19
1.8.5 Escape Sequences	1-20
1.8.6 type at Built-in Function	1-20
1.8.7 input Built-in Function	1-21
1.8.8 length of Built-in Function	1-21
1.8.9 xsize [of] Built-in Function	1-22
1.9 Match Clauses	1-23
1.10 Layout/CUA	1-24
1.10.1 CUA Defaults	1-24
1.10.2 Katakana Mnemonics	1-24
1.10.3 Keyboard Status Display Area	1-24
1.10.4 Entry Field Character Types	1-25
1.11 Library Functions	1-26
1.11.1 EsISetEgaColor() Function	1-26
1.11.2 WeekDay() Function	1-26
1.11.3 File I/O Library Subroutines	1-26
1.12 Touch Screen	1-28
1.13 Business Graphics	1-29
1.14 SQL Support	1-30
1.14.1 set Configuration Command	1-30

1.14.2 null Configuration Command	1-30
1.15 Audio Capture and Playback	1-31
1.16 APPC Support	1-32
1.16.1 Return Codes	1-32
1.16.2 APPC subroutines	1-32
1.17 3270 Support	1-34
1.17.1 Code Page	1-34
1.17.2 FIELDS.EXE	1-34
1.17.3 M3270 Module Action Routines	1-37
1.18 5250 Support	1-40
1.18.1 Code Pages	1-40
1.18.2 FIELDS52.EXE	1-40
1.18.3 M5250 Module Action Routines	1-43
1.19 3270/5250 Support	1-46
1.20 Printing and Plotting	1-47
1.20.1 plot Action Statement	1-47
1.20.2 Printing Text Files	1-47
Chapter 2. 3270/5250 Support	2-1
2.1 3270/5250 Support Differences	2-2
2.1.1 General Information	2-2
2.1.2 New Subroutines	2-3
2.1.3 Deleted Routines	2-6
2.1.4 Global Variables	2-7
2.1.5 Watch Responses	2-7
2.1.6 The Include File	2-8
2.2 Introduction	2-10
2.2.1 EASEL OS/2 EE in the 3270/5250 Environment	2-10
2.2.2 Components of 3270/5250 Support for EASEL OS/2 EE	2-10
2.2.3 Using the ECS: Required Procedures	2-11
2.2.4 Declaring the ECS Stimulus Library	2-11
2.2.5 Sample Program	2-12
2.3 3270/5250 Session and Screen Concepts	2-15
2.3.1 Session Names	2-15
2.3.2 Session States	2-15
2.3.3 Field-Oriented and Line-Oriented Screens	2-16
2.3.4 Field-Oriented (Formatted) Screens	2-16
2.4 Retrieving Field Information with ECSFIELD.EXE	2-19
2.4.1 Reading the ECSFIELD.EXE Report	2-19
2.4.2 Using ECSFIELD.EXE	2-19
2.4.3 ECSFIELD.EXE Sample Output	2-21
2.5 ECS Subroutines	2-28
2.5.1 ECS Subroutines and EASEL OS/2 EE Built-in Functions	2-29
2.5.2 Summary of ECS Subroutines	2-29
2.5.3 Connecting and Disconnecting Sessions	2-34
2.5.4 Examining the Buffer	2-40
2.5.5 Synchronizing 3270/5250 Support with the Host	2-64
2.5.6 Communicating With the Host	2-87
2.5.7 Controlling the Emulator Window	2-100
2.5.8 Miscellaneous Subroutines	2-109

2.6 Error Handling	2-119
2.6.1 Error Status Retrieval	2-119
2.6.2 Error Messages (Cross Reference)	2-120
2.7 Reserved Variables and Constants	2-129
2.7.1 Global Variables	2-129
2.7.2 Reserved Constants	2-129
2.8 Sample Program Using 3270/5250 Screens	2-132
2.8.1 Sample Program Using 3270/5250 Support	2-134
Chapter 3. Permanent and Temporary Dialogs	3-1
3.1 Overview	3-2
3.2 EASEL OS/2 EE Syntax	3-3
3.2.1 dialog box Definition	3-3
3.2.2 dialog region Definition	3-5
3.2.3 make permanent and make temporary Action Statements	3-7
3.3 Runtime Option	3-8
3.4 Layout/CUA	3-9
3.4.1 Secondary Window	3-9
3.4.2 Dialog Box	3-10
Chapter 4. Miscellaneous Enhancements	4-1
4.1 Scaling in Layout/CUA	4-2
4.2 New File I/O Error Messages	4-3
4.2.1 Open Subroutine Error Messages	4-3
4.2.2 Input Subroutines Error Messages	4-3
4.2.3 Output Subroutines Error Messages	4-4
Index	X-1

Chapter 1. DBCS Support

1.1 Overview

This chapter outlines considerations when developing and executing EASEL® OS/2® EE applications in a double-byte character set environment. The double-byte character set (DBCS) environment consists of hardware and software specifically designed to support DBCS. It is necessary for those applications written for languages which require two bytes, instead of one byte, to represent its characters. Languages which require DBCS support are Japanese, Chinese, and Korean, and thus applications written for these languages are called DBCS applications. Applications which do not use DBCS support are called single-byte character set (SBCS) applications.

The EASEL OS/2 EE DBCS Support provides support for developing and executing DBCS applications when EASEL OS/2 EE is running on OS/2 Extended Edition J (Japanese), OS/2 Extended Edition T (Traditional Chinese), and OS/2 Extended Edition K (Korean).

EASEL OS/2 EE applications can also be developed to support both SBCS and DBCS users concurrently as long as only SBCS ASCII characters are used throughout the application.

Note: EASEL OS/2 EE's DBCS Support is not an external feature of EASEL OS/2 EE as are its other support features, such as 3270 Support. Although EASEL OS/2 EE DBCS Support is referred to as a support feature, it is internal to EASEL OS/2 EE and is automatically available for use.

EASEL is a registered trademark of Easel Corporation in the United States and other countries.

OS/2 is a registered trademark of International Business Machines Corporation in the United States and other countries.

1.2 Installation

1.2.1 Machine Requirements

EASEL OS/2 EE DBCS Support prerequisites IBM DBCS OS/2, which requires that Micro Channel® architecture be supported by the programmable workstation. The IBM family 5550 S/V/T and 5570 S/V/T machines support Micro Channel architecture in the IBM PS/55 families. Therefore, EASEL OS/2 EE supports DBCS applications on any of these programmable workstations. EASEL OS/2 EE provides DBCS support for all of its features except for the following:

- Touchscreen Support
- Audio Capture and Playback Support

EASEL OS/2 EE DBCS Support does not require any additional fixed disk free space or available memory beyond that required by EASEL OS/2 EE itself.

1.2.2 Program Requirements

For DBCS language support, EASEL OS/2 EE requires that the appropriate operating system be running. See below for the operating system that corresponds to a particular DBCS language.

Japanese	IBM OS/2 Extended Edition J
Traditional Chinese	IBM OS/2 Extended Edition T
Korean	IBM OS/2 Extended Edition K

For non-DBCS language support (SBCS support only), EASEL OS/2 EE requires IBM Operating System/2™ Extended Edition.

Micro Channel is a registered trademark of International Business Machines Corporation in the United States and other countries.

Operating System/2 is a trademark of International Business Machines Corporation in the United States and other countries.

1.2.3 Device Support Limitations

EASEL OS/2 EE provides DBCS support for all of the devices supported by IBM OS/2 Extended Edition with the following limitations:

- Communications support is provided by Communications Manager, and is used for the following EASEL OS/2 EE features:
 - Asynchronous communications (ACDI)
 - 3270 and 5250 communications (EHLLAPI)
 - APPC for LU6.2 Support
- Any display connected to the 5550 S/T/V or 5570 S/V/T PS/55 family members can be utilized by EASEL OS/2 EE DBCS Support, with the following exception:
 - The PS/55 5570-S requires a display adapter for IBM DBCS OS/2, part number FEB38F7112. This display adapter is currently available in Japan only.

1.2.4 Installing EASEL OS/2 EE DBCS Support

EASEL OS/2 EE DBCS Support does not require any additional installation beyond that required by EASEL OS/2 EE itself.

When installing EASEL OS/2 EE on a DBCS environment, the following should be noted during installation:

- Before the first installation panel appears, a white screen will display momentarily. This occurs as a result of the necessary change in display modes for installation of EASEL OS/2 EE on a PS/55.
- Directory names using DBCS characters are not supported during installation.
- Use the default of N (No) for whether to install Audio Capture and Playback software. If Y (Yes) is entered, the software will be installed without error, but the feature will not work.
- Use the default of N (No) for whether to install Touchscreen software. If Y (Yes) is entered, the software will be installed without error, but the feature will not work.
- Enter Y (Yes) for whether to use a code page other than multilingual, and then enter the current process code page number or another valid DBCS code page number. Even though the multilingual code page (850) is valid for a DBCS environment, it does not provide DBCS support.

1.2.5 Files added for DBCS Support

The following three additional files are added with EASEL OS/2 EE DBCS Support.

\EASEL-EE\

DBCS.INC

Include file for DBCS support

\EASEL-EE\DLL\

ESLCUARC.DLL

National Language support for Layout/CUA™

ESLCUARE.DLL

English Layout/CUA default with provision for DBCS support.

Layout/CUA is a trademark of Easel Corporation in the United States and other countries.

1.3 Language Elements

1.3.1 Code Page Support

IBM OS/2 Extended Edition and IBM OS/2 Extended Edition J, T, and K support the following code pages for SBCS languages:

437 U.S. IBM PC code page

850 Multilingual PC code page

IBM OS/2 Extended Edition J also supports the following code pages:

932 Japanese PC code page

942 Extended Japanese PC code page (SAA)

IBM OS/2 Extended Edition T also supports the following code pages:

938 Traditional Chinese PC code page

948 Extended traditional Chinese PC code page (SAA)

IBM OS/2 Extended Edition K also supports the following code pages:

934 Korean PC code page

944 Extended Korean PC code page (SAA)

EASEL OS/2 EE allows you to compile and run your program using any of these four code pages which correspond to the operating system in use.

At compile time, EASEL OS/2 EE assumes the code page specified by the CODEPAGE configuration option in the CONFIG.ESL file. If the CODEPAGE configuration option is not specified, EASEL OS/2 EE uses the currently active code page.

When a DBCS code page is active at run time, then EASEL OS/2 EE uses the currently active code page. Even for EASEL OS/2 EE application programs compiled under a non-DBCS code page, if a DBCS code page is the currently active one, the application runs under that DBCS code page. In this case, extended ASCII characters are not displayed.

When an SBCS code page is active at run time, then the code page specified in the CONFIG.ESL file is used only for the keyboard and display code pages, and the currently active SBCS code page is used as the process code page.

To run an EASEL OS/2 EE application which was compiled under an SBCS code page, change the currently active code page to be that SBCS code page.

Note: If you are going to use communications through Communications Manager in your program, you must compile and run with the same DBCS code page specified in the CONFIG.ESL file as the one specified in the CONFIG.SYS file; otherwise, characters are not handled correctly.

1.3.2 DBCS Characters and Mixed Strings

EASEL OS/2 EE supports DBCS characters for the following language elements:

- String literals

```
"TEL:" # Double-byte alphanumeric characters
```

- Values for string constants

```
string constant Kanji is "漢字" # Double-byte Kanji characters
```

- Values for string variables

```
copy "TEL:" to Tel
```

- Path names and file names (except user-defined path names during installation)

```
"D:##顧客##住所録.txt"
```

- Comments

```
# 変更 2/28 '90 IBM太郎
```

However, DBCS characters are not supported for

- Keywords
- User-defined identifiers

Using DBCS characters for variable names may produce unexpected results.

1.3.3 Line continuation

If a string literal is too long to fit on a line, you can continue the string literal on the next line by using the EASEL OS/2 EE escape character (X'5C'), which is a backslash for multilingual and U.S., a Yen sign for Japanese, or a Won sign for Korean, by itself at the end of the line. For example:

```
copy "this line continues ¥  
to next line"  
to LongString
```

It is also possible to specify multiple string literals in many EASEL OS/2 EE statements. The following example and the previous example will have exactly the same results.

```
copy "this line continues "  
    "to next line"  
    to LongString
```

Note: Comments following a line continuation on the same line cause a compilation error. The following example would cause an error.

```
copy "this line continues ¥ # line continuation  
    to next line"  
    to LongString
```

1.3.4 Escape Sequences in String Literals.

In a string literal, you can use the EASEL OS/2 EE escape character (X'5C'), which is a backslash for multilingual and U.S., a Yen sign for Japanese, or a Won sign for Korean, followed by a code to specify special control characters.

The Yen sign and Won sign are also used as currency symbols. However, at compile time EASEL OS/2 EE parses these symbols in string literals as the EASEL OS/2 EE escape character, NOT as currency symbols. For example, in

```
copy "¥5000.yen" to GDataY[1,2]
```

the ¥ character is interpreted as the EASEL OS/2 EE escape character, not as a currency symbol.

To use the EASEL OS/2 EE escape character as a character symbol, use two EASEL OS/2 EE escape characters together, which is ¥¥ for Japanese. For example:

```
copy "¥¥5000.yen" to GDataY[1,2]
```

1.3.5 Katakana Mnemonics

EASEL OS/2 EE supports Katakana Mnemonics (applicable only for Japanese) for action bar pulldowns and pulldown choices in the same way as in the resource files of Presentation Manager™ (PM) applications. The mnemonics are specified by using the EASEL OS/2 EE escape character, followed by the octal code, followed by the appropriate mnemonic character.

Presentation Manager is a trademark of International Business Machines Corporation in the United States and other countries.

In the following example of an action bar template definition, the character following octal code 036 represent the alphabetic mnemonics and the character following octal code 037 represents the Katakana mnemonics.

```
action bar PrimaryWindowABCUA is
  pulldown File text  "ファイル(¥036F¥037フ)"
    choice New  text  "新規作成(¥036N¥037サ)"
    choice Open text  "オープン(¥036O¥037ト)"
  end pulldown
end action bar
```

You can enter these octal codes while using Layout/CUA or by editing the EASEL OS/2 EE source program using a text editor. Katakana and Alphabetic mnemonics can be switched to each other at run time by using the PM Control Panel. It is recommended that users specify both alphabetic and Katakana mnemonics.

The PM mnemonic symbol (X'7E'), which is a tilde for multilingual and U.S., or an overscore for DBCS languages, can be used to specify alphabetic mnemonics as well. However, you must use either the PM mnemonic symbol or the octal code in the text string, but not both. Therefore, the following example would cause a compilation error.

```
choice New text " ~New (¥036W¥037サ)"
```

If the PM mnemonic symbol is placed in front of a DBCS character, it does not work as a mnemonic.

1.4 extract from Action Statement

With the **extract from** action statement you can extract data fields from a string value. For example, data can be extracted from the input from the keyboard or from a textual region and then copied to specified variable(s).

In the following TAKE clauses and SKIP clauses a DBCS blank is regarded as a non-printing character.

- **take [by] word** and **skip [by] word**
- **take [by] number** and **skip [by] number**
- **take [by] float** and **skip [by] float**
- **take to word** and **skip to word**
- **take to number** and **skip to number**
- **take to float** and **skip to float**

For example, when EASEL OS/2 EE executes **skip to word** for the input:

```
TEL_x___03-771-1151
```

and the pointer is positioned as indicated. The pointer skips the three DBCS blanks and moves to the position as shown below.

```
TEL____03-771-1151
```

If **take to word** Blank is executed for the above example, the three DBCS blanks would be taken and copied to the variable Blank.

For the **take to word** clause, if the current pointer is located at the second byte of a DBCS printing character, the pointer remains at the same position, as shown below, and returns a null string .

```
TEL_x___03-771-1151
```

For the **take [by] word** clause, if the current pointer is located at the second byte of a DBCS printing character, it takes up to, but not including, the first non-printing character, and the first fragment byte is copied as an SBCS blank.

If you specify column position as:

- **take [by] COLUMN** or **skip [by] COLUMN**
- **take to COLUMN** and **skip to COLUMN**

EASEL OS/2 EE adheres to the column position specified in COLUMN. If the current cursor position is at the second byte of a DBCS character or if the specified COLUMN causes the TAKE clause to copy only the first byte of a DBCS character, the fragment byte is replaced with an SBCS blank.

1.5 Fonts

1.5.1 Installation of Resolution Dependent Fonts

For the PS/55 display adapter, whose resolution is 1040x768, EASEL OS/2 EE uses resolution-dependent fonts of resolution 1024x768.

1.5.2 Default Font

If no font is specified in the environment declaration, the font **medium** is used as the default font when a SBCS code page is active at run time. When a DBCS code page is active at run time, the default font is the PM font MINCHO for OS/2 EE J, and GOTHIC for OS/2 EE T and K. Although fonts MINCHO and GOTHIC are not exactly the same as font **medium**, they are chosen as the default, since they are the closest to, but smaller than, the font **medium**.

1.5.3 OS/2 Presentation Manager Fonts

The PM fonts Courier, Helv, System Proportional, System Monospaced, and Tms Rmn are available as code page 850 fonts.

Additional PM fonts available in OS/2 EE J are:

Facename	Type	Code Page	Available Sizes
MINCHO	fixed-width	932, 942, 850 932, 942	8x16, 12x24, Outline ¹ 16x32 ¹
MINCHO DRAFT²	fixed-width	932, 942	6x12, 7x14, 8x16, 9x18, 10x20, 11x22, 16x32, 18x36, 24x48
GOTHIC²	fixed-width	932, 942	12x24, 16x32, Outline ¹

¹ These fonts are not available in the OS/2 Extended Edition packages but are available as a separate package, OS/2 Extended Edition Outline Fonts.

² These fonts are provided with OS/2 Extended Edition J, T, and K as an optional feature.

Additional PM fonts available in OS/2 EE T are:

Facename	Type	Code Page	Available Sizes
MINCHO	fixed-width	938, 948, 850 938, 948	12x24 Outline ¹
GOTHIC	fixed-width	938, 948, 850	8x16

Additional PM fonts available in OS/2 EE K are:

Facename	Type	Code Page	Available Sizes
MINCHO	fixed-width	934, 944, 850 934, 944	12x24 Outline ¹
GOTHIC	fixed-width	934, 944, 850	8x16

To use fonts that support DBCS characters, use the **font is** font definition. For example:

```
font TextFont is          # text font definition
    facename "MINCHO"
    size 12 24
```

```
font TitleFont is        # header text font definition
    facename "MINCHO"
    size 40 20
    bold outline
```

```
font "TextFont"          # font reference (environment declaration)
```

If the specified facename is not found, then the default font is used, and a warning is generated during runtime.

If the font specified in a font reference is not found:

- Check whether the FONTPATH configuration option is correctly specified in the CONFIG.ESL file.
- Check if the font is registered by using the PM Control Panel. You can use the NLSSAMPL program provided by the OS/2 Toolkit to identify currently available fonts and their sizes.

¹ These fonts are not available in the OS/2 Extended Edition packages but are available as a separate package, OS/2 Extended Edition Outline Fonts.

1.5.4 EASEL OS/2 EE Fonts

EASEL OS/2 EE has four types of resolution dependent fonts: **medium**, **large**, **mcaps**, and **lcaps**.

When a DBCS code page is active, fonts **medium** and **large** are replaced with PM fonts that are closest to, but smaller than, these. For example, if you specify font "medium", a PM font is used to display the characters, which is the same as the default font. If you specify font "large", the font used is the PM font you see in PM windows.

Fonts **mcaps** and **lcaps** are used when you display capital letters. However, these fonts do not support any DBCS characters.

EASEL OS/2 EE fixed cell fonts do not support DBCS characters.

1.6 Keyboard Status Display Area

With a Keyboard Status Display Area, users can identify the current keyboard status and can enter DBCS characters. The keyboard status display area can be added to the bottom of the following objects:

- Dialog box
- Dialog region
- Textual region
- Graphical region
- Image region

To define the Keyboard Status Display Area in object definitions, use the **keyboard status area** optional attribute shown below.

Model

[NORMAL_REGION_DEFINITION]

[**keyboard status area**]

You can define the Keyboard Status Display Area in an object using the object information dialog boxes of Layout/CUA. The Keyboard Status Display Area is displayed by default when objects are created.

If you run EASEL OS/2 EE under code pages 850 or 437, PM simply ignores this attribute if it is specified.

1.7 Entry Field Data Types

Users can select the type of data to be entered into single-line entry fields. EASEL OS/2 EE supports the following four entry fields data types:

char	For SBCS only
dbcs	For DBCS only
any	For DBCS and SBCS characters mixed
mixed	For DBCS and SBCS characters mixed (used for exchanging data between ASCII systems and EBCDIC systems)

To specify the data type in **entry field** definitions, use the **datatype** is optional attribute shown below.

Model

[ENTRY_FIELD_DEFINITION]

[datatype is { char | ducs | mixed | any }]

If a data type is not specified, the type **any** is the default.

If the entry field is to be used with communications, it is recommended that the **mixed** data type is used, since the string will go through ASCII to EBCDIC and EBCDIC to ASCII conversions.

In Layout/CUA, the data type may be selected from the Entry Field Information dialog box.

These data types are supported only for DBCS code pages. If you run EASEL OS/2 EE under code pages 850 or 437, PM simply ignores this attribute if it is specified.

Note: Entry field data types are not supported in **multiline entry field** definitions or for the entry field portion in **combination box** definitions.

1.8 Textual Regions

A textual region is used for displaying and manipulating text data from the keyboard, text files, local applications and host screens through the communication utilities in EASEL OS/2 EE.

A textual region could also be used as a text editor by directing keyboard input to the textual region. In many editors that handle DBCS characters, the cursor and column position is always adjusted to be located at the first byte of DBCS characters. For example, when moving the cursor to the left or right on a line of DBCS characters, the cursor moves by two columns, or two bytes, automatically. In many EASEL OS/2 EE applications, however, keeping track of user specified column positions is important. For example, EASEL OS/2 EE can invoke an action as a response to finding a character in a specified position, or locating a string in a particular position on a host screen. In order to keep this feature, EASEL OS/2 EE does *not* adjust the column and cursor position.

1.8.1 Moving the Text Cursor and Specifying Column Position

When using statements which move the cursor position in a textual region, you must remember that the cursor position is not adjusted to the first byte of DBCS characters. As an example, if a DBCS character is located in columns 4 and 5, on line 10, the drawing statement `move to column 5 line 10` would cause the cursor to move to the second byte (right-half portion) of the DBCS character.

Statements which could result in a cursor position other than the first byte of a DBCS character are:

- **move** { [**by**] ... | **to** ... }
- **find in** ... **from** [**column**] ...
- **add to** ... **at** [**position**] **column** ...
- **xcoord**
- **insert** [**at cursor** | **after cursor**] ...
- **overwrite** [**at cursor**] ...

The built-in function **type at**, described in 1.8.6, "type at Built-in Function" on page 1-20, can be used to query the character type (that is, SBCS character, or the first or second byte of a DBCS character) at the current cursor position, or at any other specified position.

1.8.2 Block Operations

Because EASEL OS/2 EE is designed to keep application-specified cursor positions rather than adjust to the first byte of DBCS characters when moving the cursor, DBCS characters could be bisected by the following block operations:

- **insert block ...**
- **overwrite block ...**
- **blank block ...**
- **remove block ...**
- **textual block ...**

For these block operations, EASEL OS/2 EE does not adjust the block boundary, but keeps the block as a rectangle and replaces the fragment byte of a bisected DBCS character with an SBCS blank.

For example, when using the **insert block** statement:

```
response to CopyText
  add to NewText
    insert block column 3 line 1
                thru column 8 line 3
                from OldText
```

If the second byte of a DBCS character in OldText is located in column 3 on lines 1, 2, or 3, it is a fragment byte, and that byte is displayed as an SBCS blank. If the first byte of a DBCS character in OldText is located in column 8 on lines 1, 2, or 3, it is also a fragment byte, and that byte is also displayed as an SBCS blank. Although the text from OldText changes once inserted into object NewText, it remains the same in object OldText.

Bisection may also occur in NewText. If the second byte of a DBCS character is at the cursor position where the text will be inserted, the inserted text will separate the first and second bytes. EASEL OS/2 EE replaces the fragment bytes with SBCS blanks.

The **textual block** built-in function returns the contents of a specified block with the fragment bytes replaced by SBCS blanks (spaces).

The same treatment for fragment bytes of DBCS characters applies to the following 3270 and 5250 action routines:

- **ReadScreen**
- **ReadColoredScreen**
- **M52_ReadScreen**
- **M52_ReadColoredScreen**

Exceptions to the fragmentation are:

- **make ... block ... [color] COLOR**
- **COLOR_SPECIFICATION** (when specifying text colors in an ASCII text file for use with the **CText** module)

For these block operations, DBCS characters with the first or second byte outside the block boundary are colored so both bytes have the same color as the byte which is inside the block boundary.

1.8.3 Segment Operations

For segment operations, as with block operations, EASEL OS/2 EE does not adjust the segment boundary, but keeps the segment as its original shape and replaces the fragment byte of a bisected DBCS character with an SBCS blank. DBCS characters can be bisected by the following segment operations:

- **insert segment ...**
- **overwrite segment ...**
- **remove segment ...**
- **textual segment ...**

Exceptions to the fragmentation are:

- **emphasize segment ...**
- **make ... segment ... [color] COLOR**
- **COLOR_SPECIFICATION** (when specifying text color in an ASCII text file for use with the **CText** module)

For these segment operations, DBCS characters with the first or second byte outside the segment boundary are colored so both bytes have the same color as the byte which is inside the segment boundary.

1.8.4 String Operations

For string operations, as with segment and block operations, EASEL OS/2 EE does not adjust the string, but keeps the string as it is and replaces the fragment byte of a bisected DBCS character with an SBCS blank. But, in this case, bisection can only occur to the DBCS characters in the text where the operation is to be executed. The following string operations apply:

- **insert ... TEXT_STRING**
- **overwrite ... TEXT_STRING**

1.8.5 Escape Sequences

Users can use **insert** and **overwrite** with string literals that contain EASEL OS/2 EE escape sequences. EASEL OS/2 EE handles the bisection of DBCS characters from **insert** and **overwrite** when used with the tab, backspace and newline EASEL OS/2 EE escape sequences as follows:

insert a tab (\t)

When inserting a tab at the second byte of a DBCS character, both the first byte and the second byte of the bisected DBCS character are replaced with SBCS blanks.

overwrite with a tab (\t)

When overwriting with a tab at the second byte of a DBCS character, the first byte of the DBCS character is replaced by an SBCS blank. When the tab-stop lies on the first byte of a DBCS character, the second byte of the character is replaced with an SBCS blank.

insert or **overwrite** with a backspace (\b)

When inserting or overwriting with a backspace at the position where the character to the left is a DBCS character, the second byte of the DBCS character is deleted and the first byte is replaced with an SBCS blank. If the cursor is at the second byte of a DBCS character, the first byte of the character is deleted and the second byte is replaced with an SBCS blank.

insert a newline (\n)

When inserting a newline at the second byte of a DBCS character, the first byte of the character is replaced with an SBCS blank and the first column of the next line is an SBCS blank.

1.8.6 type at Built-in Function

Because EASEL OS/2 EE does not adjust the cursor or column position to the first byte of a DBCS character, you may need to check which type of byte is located at the cursor position or a position specified by the column and line number of a textual region. For this purpose you can use the **type at** built-in function.

Model

```
type at { cursor | [column] COL_NO [line] LINE_NO }  
      char of TR_NAME
```

In the model, COL_NO and LINE_NO are the column and line number, respectively, within the textual region, and TR_NAME is the identifier for the textual region.

This function can be used only with textual regions.

The value returned is an integer as shown below.

Character Type	Return Value
SBCS	SBCS
First byte of DBCS character	DBCS1
Second byte of DBCS character	DBCS2

If no character is found at the specified position, a negative value (-1) is returned.

The SBCS, DBCS1, and DBCS2 identifiers are defined as integer constants in the include file DBCS.INC. They could also be defined as constants in your EASEL OS/2 EE application, if the DBCS.INC file is not included.

The following example adjusts the cursor position to the first byte of a DBCS character.

```
if (type at cursor char of Textual = DBCS2) then
    add to Textual
        move -1 0
end if
```

1.8.7 input Built-in Function

The **input** built-in function returns a string value received from an application or from the keyboard for **response to char** or **response to line** statements.

The **response to char** statement responds to any character. Therefore, if the first character is a DBCS character, the **input** function returns the DBCS character.

1.8.8 length of Built-in Function

The **length of** built-in function returns the number of *bytes* in a specified string variable. In the first example below, the value returned is 3, while the value returned is 6 in the second example.

```
copy "TEL" to Tel
if (length of Tel > 3) then
    :
end if
```

```
copy "TEL" to Tel
if (length of Tel < 7) then
    :
end if
```

1.8.9 **xsize [of]** Built-in Function

The **xsize [of]** built-in function represents the X dimension of the textual region's size, in its parent's coordinate system. For textual regions defined by **window size**, this is the number of columns, multiplied by the column size, in the textual region's coordinates, when an application program runs under a DBCS code page.

1.9 Match Clauses

Match clauses are used in **response to char**, **response to line**, and **switch** statements.

In a match clause, shown below, a DBCS blank is regarded as two SBCS blanks.

```
[col NUM1 [thru NUM2] ] { word | number | blank }
```

Thus, the **response statement**, response to line word from keyboard, is not invoked when only DBCS blanks are typed in, while the response statement, response to char blank from keyboard is invoked for a DBCS blank, and the input function returns a DBCS blank.

When you specify

```
response to char  
col 2 thru 5 blank from keyboard
```

and type in a DBCS blank at column 1, the response is invoked, and the **input** function returns a DBCS blank. If you type in abcd followed by a DBCS blank, the response is invoked, and the **input** function returns a string containing abcd and an SBCS blank.

When you specify

```
response to char  
col 2 thru 5 word from keyboard
```

and type in only one DBCS printing character at column 1 or four blanks followed by a DBCS printing character, the response is not invoked. But, if any DBCS character is typed in between columns 2 and 5, so that none are bisected, the response is invoked, and the **input** function returns the string up to the character typed.

For match clauses with the **switch** action statement, the same process for matching applies.

1.10 Layout/CUA

1.10.1 CUA Defaults

Primary and secondary windows have default action bars and pulldown choices. These are translated into Japanese when using OS/2 EE J with an active code page of 932 or 942. To develop DBCS support applications with English defaults, the **-eng** command line option needs to be specified with ESLCUA. The following example shows how to use this option.

```
esl\cua -eng
```

The default action bars and pulldown choices are not translated for Traditional Chinese or Korean, and are displayed in English when running EASEL OS/2 EE under OS/2 EE T and K. To display the translated defaults, the file ESLCUARC.DLL must exist in the DLL sub-directory of the EASEL-EE directory.

When you run Layout/CUA under an SBCS code page (e.g. 850 or 437), English defaults will be displayed.

1.10.2 Katakana Mnemonics

In the Action Bar Choice Info and Pulldown Choice Info dialog boxes, you can define Katakana mnemonics (for Japanese only) and alphabetic mnemonics in the form of octal codes as described in 1.3.5, "Katakana Mnemonics" on page 1-8. The PM mnemonic symbol (X'7E'), a tilde for multilingual and U.S., or an overscore for DBCS languages, can be used as well for alphabetic mnemonics.

A mnemonic is not allowed to be used twice in the same action bar or in the same pulldown. If you happen to specify the same mnemonic, Layout/CUA generates a warning.

Note: If the active code page is not one of the Japanese code pages, Katakana mnemonics will not work.

1.10.3 Keyboard Status Display Area

When you create a primary window, secondary windows, or dialog boxes for your interface, they will appear with a Keyboard Status Display Area as one of the default attributes of the object.

If you do not need the Keyboard Status Display Area for a particular object, you can choose the No Keyboard Shift Status Area check box in the object's information dialog box. When you check it, the window will no longer have a Keyboard Status Display Area.

When you generate EASEL OS/2 EE code using the Convert choice in the File pulldown, the objects that have a Keyboard Status Display Area will include the keywords **keyboard status area** in their definition.

Note: If you run Layout/CUA under an SBCS code page, the Keyboard Status Display Area does not appear at all.

1.10.4 Entry Field Character Types

The Entry Field Information dialog box has radio buttons for specifying one of four allowable character types:

- Char
- DBCS
- Mixed
- Any

The default character type is Any. If you choose type Char, the text that is to be displayed as the initial text of the entry field accepts only SBCS characters. When type DBCS is chosen, the entry field accepts only DBCS characters.

1.11 Library Functions

1.11.1 EsiSetEgaColor() Function

The **EsiSetEgaColor()** function can be used to set colors for a fullscreen EASEL OS/2 EE program. When EASEL OS/2 EE is running on a PS/55, the default colors for a fullscreen program are the same as those of a windowed program.

1.11.2 WeekDay() Function

The **WeekDay()** function does not provide DBCS support.

1.11.3 File I/O Library Subroutines

1.11.3.1 DBCS Field in a record Definition

A DBCS field type that accepts only DBCS characters is supported. To specify this DBCS field in a record definition, use the following model.

Model

VARIABLE_NAME VAR_WIDTH, **dbcs**

Where VARIABLE_NAME is a string variable, and VAR_WIDTH is the number of bytes in the string and must be an even number. Specifying an odd number for VAR_WIDTH generates a compiler error.

The DBCS field is supported only for File I/O library subroutines.

If you do not specify **dbcs**, the field type is interpreted as type Any (that is, mixture of SBCS and DBCS characters). If **dbcs** is added to any field type other than a string variable, it is ignored.

You can put only a DBCS character string into a DBCS field. If SBCS characters are contained in a DBCS field, the **errorlevel** built-in function will be set after calling any I/O functions:

errorlevel	Message
30	Invalid character in DBCS field

When the length of a DBCS character string is less than the specified length of the DBCS field, the remaining columns will be padded by DBCS blanks. If the length is longer, the string will be truncated and trailing characters will be ignored.

1.11.3.2 Bisection in reading/writing

When reading data, bisection of DBCS characters may occur due to field specification errors or the wrong data is in the file being read. When this happens in reading, the following **errorlevel** will be set.

errorlevel	Message
31	Last DBCS character has been bisected in reading

Bisection of a DBCS character could occur at the end of a string field also when writing the string. In this case the fragment byte of a DBCS character is replaced with an SBCS blank without a warning or error.

1.12 Touch Screen

EASEL OS/2 EE touch screen does not provide DBCS support.

1.13 Business Graphics

In business graphics, data elements are assigned to **GDataY** and **GDataX** in the form of strings, in which users can use the dollar sign as well as commas, decimal points, fractions, and parentheses. However, the corresponding currency signs, Yen sign for Japanese and Won sign for Korean, in string literals are used to represent the EASEL OS/2 EE escape character in compilation.

For example, in the statement

```
copy " ¥5000 " to GDataY[10,12]
```

the ¥ character is not interpreted as a Yen sign but as the EASEL OS/2 EE escape character and the number 5000 is not copied into the array **GDataY**. Instead, ¥5000 is interpreted as an octal code and the last 0 is copied into **GDataY**.

To use the EASEL OS/2 EE escape character as a character symbol, use two EASEL OS/2 EE escape characters together, which is ¥¥ for Japanese.

Note: When a string that contains a character symbol, for example a ¥ , is copied into **GDataY** or **GDataX** at run time, that is, when reading data files for graphics presentation, the character symbol is stripped from the digits that follow and the correct number is copied into the array.

1.14 SQL Support

1.14.1 set Configuration Command

The **set** COLUMN_NO WIDTH command sets the display width for the specified column. For DBCS fields (graphic SQL types) the field width WIDTH is the number of bytes, as it is for other types of fields. (In Data Base Manager, it is given in character units.) When you inquire about the width of a graphic field with the **widths** SELECT_STATEMENT command, the output width returned is the maximum number of bytes.

The width of a column in which you put DBCS data should be an even number. When a DBCS field is stored into a column with an odd numbered width, the width is reduced automatically by one byte (column). When the length of data is longer than a specified width, the data is truncated. If it is shorter, then the data is padded with DBCS blanks (0x81,0x40) except for the null indicator field as described below.

1.14.2 null Configuration Command

The **null** [STRING] command sets the null indicator string which is returned when the database contains no data in a particular row and column. You can specify STRING to be of mixed type, even for a graphic (DBCS) field. Because of this, data in the null indicator string is padded by SBCS blanks even if it is a graphic (DBCS) field.

1.15 Audio Capture and Playback

Audio Capture and Playback does not provide DBCS support.

1.16 APPC Support

1.16.1 Return Codes

The following EASEL OS/2 EE APPC integer constants and their corresponding return code values pertain to DBCS character strings:

errorlevel	Error Integer Constant/Explanation
61	APPC_ERR_TRANSLATED_STR_TOO_BIG Translated string is too large.
62	APPC_ERR_INV_DBCS_START_INDEX Invalid DBCS start index value.
63	APPC_ERR_INV_DBCS_END_INDEX Invalid DBCS end index value.
64	APPC_ERR_INV_DBCS_STRING Invalid DBCS string.
65	APPC_ERR_INV_TRANSLATE_TABLE Invalid Translation table value.

1.16.2 APPC subroutines

The following subroutines can be used to convert DBCS, SBCS, and mixed SBCS and DBCS strings:

- **APPCSendString()** with **APPC_Convert** specified as the **AUTO_CONVERT** argument
- **APPCGetString()** with **APPC_Convert** specified as the **AUTO_CONVERT** argument
- **APPCConvertA2E()**
- **APPCConvertE2A()**

DBCS characters will only convert successfully on a system that supports a DBCS code page and that code page must be specified in the CONFIG.SYS file. EBCDIC DBCS characters are surrounded by starting and ending delimiters, SO (shift out) and SI (shift in) characters, respectively.

The following restrictions apply to the index values that may be specified with the **APPCCConvertA2E()** and **APPCCConvertE2A()** subroutines when a string contains DBCS characters.

- **APPCCConvertA2E()**

The start index value must indicate the position of an SBCS character or the first byte of a DBCS character.

The end index value must indicate the position of an SBCS character or the second byte of a DBCS character.

- **APPCCConvertE2A()**

The start index value must *not* indicate a position within a DBCS delimiter enclosed string. It may indicate the position of the starting delimiter, SO (X'0E').

The end index value must *not* indicate a position within a DBCS delimiter enclosed string. It may indicate the position of the ending delimiter, SI (X'0F').

Note: When converting a DBCS string from ASCII to EBCDIC, the size of the buffer for the EBCDIC string is double that of the ASCII string to allow for the maximum number of SO and SI bytes.

1.17 3270 Support

1.17.1 Code Page

All host code pages that are supported by Communications Manager for 3270 emulation are supported. This is true for OS/2 EE J, T, and K.

Although the code page under which you run EASEL OS/2 EE can be any one of the code pages that OS/2 provides, EHLLAPI uses the primary code page specified in the CONFIG.SYS file for conversion of data from ASCII to EBCDIC.

1.17.2 FIELDS.EXE

With the FIELDS.EXE utility you can get information on the current screen, on each field in the screen, and the full contents in each field. EASEL OS/2 EE DBCS Support provides additional FIELDS functions that are unique for the DBCS environment.

1.17.2.1 SO/SI Printing Option

The 3270 host represents DBCS strings by enclosing them with the SO (shift out) and SI (shift in) characters. Their hexadecimal values are X'0E' and X'0F', respectively, in the EBCDIC code page.

You can choose to print the SO and SI characters, either as blanks or as "<" and ">" with the command option **-sosi**. The option parameter **-sosi** specifies to print the SO and SI character as "<" and ">" respectively. By default, the SO and SI characters are printed as SBCS blanks.

1.17.2.2 DBCS character at column 80

Normally a display shows 80 columns on one line. But when handling DBCS characters, the first byte of a DBCS character may be located at column 80. In this case, the second byte of the DBCS character is put at column 81 on the same line, and the first column of the next line is printed as a blank.

1.17.2.3 Field Summary

The 3270 emulator supports four types of fields:

Alphanumeric	For SBCS alphanumeric characters only
Numeric	For SBCS numeric characters only

DBCS

For DBCS characters only

Mixed

For DBCS and SBCS characters

Information on the type of each field is given in column Type in the field summary as shown below. Column Type replaces column Num of previous releases of EASEL OS/2 EE.

The last column, "First 26 Columns", displays the first 26 bytes of each field. (Descriptions for the FIELDS.EXE output using the term "characters" should be interpreted as "bytes"). When the 26th column contains the first byte of a DBCS character, 27 columns are displayed to include the whole DBCS character.

1.17.2.4 Sample Field Summary Output

Screen contains 39 fields.
Cursor is at [11,15].

Field summary:

Unp = Unprotected	AN = Alphanumeric	NU = Numeric
Nds = Nondisplay	DB = DBCS	MX = Mixed
Int = Intensified		
Mod = Modified		

Fld#	Start	End	Length	Unp	Nds	Int	Mod	Type	First 26 Columns
1	[1,2]	[1,80]	79					MX	[
2	[2,2]	[2,80]	79		X			MX	[
3	[3,2]	[3,80]	79					MX	[
4	[4,2]	[4,80]	79					MX	[
5	[5,2]	[5,13]	12		X			MX	[DEFAULT ==>]
6	[5,15]	[5,22]	8	X		X	X	MX	[兩方 OK]
7	[5,24]	[5,80]	57			X		MX	[
8	[6,2]	[6,80]	79					MX	[
9	[7,2]	[7,13]	12			X		MX	[PUREDBCS ==>]
10	[7,15]	[7,22]	8	X		X	X	DB	[漢字]
11	[7,24]	[7,80]	57			X		MX	[
12	[8,2]	[8,80]	79					MX	[
13	[9,2]	[9,13]	12			X		MX	[SBCSONLY ==>]
14	[9,15]	[9,29]	15	X		X	X	AN	[ABCD]
15	[9,31]	[9,80]	50			X		MX	[
16	[10,2]	[10,80]	79					MX	[
17	[11,2]	[11,13]	12			X		MX	[NUMERIC ==>]
18	[11,15]	[11,22]	8	X		X		MX	[]
19	[11,24]	[11,80]	57			X		MX	[
20	[12,2]	[12,80]	79					MX	[
21	[13,2]	[13,80]	79			X		MX	[
22	[14,2]	[14,53]	52			X		MX	[
23	[14,55]	[14,71]	17					MX	[Mix Wrapped ==>]
24	[14,73]	[15,22]	30	X		X	X	MX	[1234567890
25	[15,24]	[15,53]	30					MX	[
26	[15,55]	[15,71]	17					MX	[Mix Wrapped ==>]
27	[15,73]	[16,22]	30	X		X	X	MX	[S 1234567890
28	[16,24]	[16,80]	57					MX	[
29	[17,2]	[17,80]	79					MX	[
30	[18,2]	[18,18]	17					MX	[Mix Tranctd ==>]
31	[18,20]	[18,28]	9	X		X		AN	[..a.b.c.d]
32	[18,30]	[18,80]	51					MX	[
33	[19,2]	[19,80]	79					MX	[
34	[20,2]	[20,80]	79					MX	[
35	[21,2]	[21,80]	79					MX	[
36	[22,2]	[22,14]	13			X		MX	[WRAPPED FIELD]
37	[22,16]	[22,17]	2					MX	[]
38	[22,19]	[24,61]	203	X		X		MX	[ABCDEFGHIJKLM
39	[24,63]	[24,80]	18					MX	[]

1.17.3 M3270 Module Action Routines

1.17.3.1 action Init3270 and action Connect

Action routines **Init3270** and **Connect** have the following additional input variables in addition to the input variable **SessionName**.

<i>Input</i>	DelSOSI	(boolean)
<i>Variables:</i>	SO	(string)
	SI	(string)
	CaseInsensitive	(boolean)

Remarks: The boolean variable **DelSOSI**, when set to **true**, discards the SO and SI characters when reading host screen data into EASEL OS/2 EE objects. If set to **false**, the default value, the SO and SI characters remain and are translated to the string value of the input variables **SO** and **SI** respectively. The values of **SO** and **SI** must be assigned values which are 0 or 1 byte in length each. If they are not initialized, they are treated as SBCS blanks.

The **DelSOSI** option affects only functions that read host screens such as **ReadField**, **ReadLine**, **ReadScreen** and **ReadColoredScreen**, not functions which write to host screens.

CaseInsensitive, when set to **true**, executes read and watch actions as case-insensitive. The default value is **false**, case-sensitive.

Example:

```
response to Start3270
  copy "A" to SessionName
  copy "<" to SO           # display SO as "<"
  copy ">" to SI           # display SI as ">"
  copy true to CaseInsensitive # case-insensitive
  action Init3270
```

The values of these variables are used in subsequent **Connect** action routines. For example, if you issue action **Connect** after the above example without specifying **SessionName**, then **Connect** tries to connect to Session A. The **SO** and **SI** variable values set by **Init3270** are kept as the default values and may be used for all sessions. To use different variable values for a specific session, copy the new values to the variable, and then connect to that session. These new values then become the new defaults. To reset the values to their original values, the **copy** statement can be used anytime following **Connect**. For example:

```
copy "B" to SessionName      # connect session B
copy false to CaseInsensitive # only for session B
action Connect
copy true to CaseInsensitive  # reset to previous value
```

Note: If you omit the last line copy true to CaseInsensitive, all **Connect** actions thereafter use the value **false** for **CaseInsensitive** unless a different value is specified explicitly.

1.17.3.2 action GetFieldAttributes

GetFieldAttributes has one additional output variable which contains the type of the field.

<i>Output Variables:</i>	FieldType	(integer)
<i>Remarks:</i>	GetFieldAttributes sets FieldType to one of the following constants:	
	FT_ALPHANUMERIC	For an SBCS alphanumeric field
	FT_NUMERIC	For an SBCS numeric field
	FT_DBCS	For a DBCS field
	FT_MIXED	For a Mixed SBCS/DBCS field
	If the field type is FT_ALPHANUMERIC or FT_NUMERIC , then the output variable Alphanumeric is set to true . The above constants are defined in the DBCS.INC include file.	

1.17.3.3 Bisected DBCS Characters

In read actions **ReadLine**, **ReadScreen** and **ReadColoredScreen**, DBCS characters can be bisected at the borders of the EASEL OS/2 EE areas holding the host data. If this happens, EASEL OS/2 EE replaces the fragment bytes of bisected DBCS characters with SBCS blanks.

1.17.3.4 Watch Action Routines

Watch actions **WatchForChar** and **WatchForNotChar** support watching for DBCS characters. For example the following watch definition is valid.

```
action Sequence_4 is
  action DefineWatch
  copy 20 to WatchRow
  copy 12 to WatchCol
  copy "𐀀" to WatchChar
  action WatchForChar
```

When a DBCS character is being watched, the variable **WatchCol** signifies the first byte of the DBCS character.

1.17.3.5 action FindLastNonBlankLine

FindLastNonBlankLine returns the number of the last non-blank line on the screen preceding the line on which the cursor lies. With this action you can skip blank lines to find, for example, the most recent prompt or message.

FindLastNonBlankline regards DBCS blanks as blanks, as well.

1.17.3.6 action EmulateAndWatch

DBCS characters are supported in the MATCH_STRING clause of screen ID files.

1.17.3.7 action WriteField

WriteField supports mixed strings for the input variable **FieldText**. **FieldText** can be longer than the specified length of **FieldNumber** as a result of the insertion of SO and SI characters. In this case, trailing characters are truncated and an error is returned to the application. Errors are returned only when truncation occurs due to insertion of SO and SI characters.

1.17.3.8 action Print3270Screen

Print3270Screen has an input variable to specify how to print SO and SI characters.

<i>Input</i>	PrintSOSI	(boolean)
<i>Variables:</i>		

<i>Remarks:</i>	If PrintSOSI is set to true , SO and SI characters are printed as “<” and “>” respectively. If set to false , the default value, SO and SI characters are printed as SBCS blanks.
-----------------	--

1.18 5250 Support

1.18.1 Code Pages

All host code pages that are supported by Communications Manager for 5250 emulation are supported. This is true for OS/2 EE J, T, and K.

Although the code page under which you run EASEL OS/2 EE can be any one of the code pages that OS/2 provides, EHLLAPI uses the primary code page specified in the CONFIG.SYS file for conversion of data from ASCII to EBCDIC.

1.18.2 FIELDS52.EXE

EASEL OS/2 EE DBCS Support provides additional FIELDS52 functions that are unique for the DBCS environment.

1.18.2.1 SO/SI Printing Option

The 5250 host represents DBCS strings by enclosing them with the SO (shift out) and SI (shift in) characters. Their hexadecimal values are X'0E' and X'0F', respectively, in the EBCDIC code page.

You can choose to print the SO and SI characters, either as blanks or as "<" and ">" with the command option **-sosi**. The option parameter **-sosi** specifies to print the SO and SI characters as "<" and ">" respectively. By default, the SO and SI characters are printed as SBCS blanks.

1.18.2.2 DBCS character at column 80

Normally a display shows 80 columns on one line. But when handling DBCS characters, the first byte of a DBCS character may be located at column 80. In this case, the second byte of the DBCS character is put at column 81 on the same line, and the first column of the next line is printed as a blank.

1.18.2.3 Field Summary

The 5250 emulator supports the following types of fields specifically for DBCS characters.

Katakana	For Katakana characters only
DBCS Only	For DBCS characters only
Open	For DBCS and SBCS characters

Either For DBCS only or SBCS only characters (decided by the first character entered)

Information on the type of each field is given in column Type in the field summary as shown below.

The last column, "First 26 Columns", displays the first 26 bytes of each field. When the 26th column contains the first byte of a DBCS character, 27 columns are displayed to include the whole DBCS character.

1.18.2.4 Sample Field Summary Output

Screen contains 47 fields.

Cursor is at [7,20].

Field summary:

Unp = Unprotected	AN = Alphanumeric	DO = Digits only
Nds = Nondisplay	AB = Alphabetic only	MS = Mag stripe data only
Int = Intensified	NS = Numeric shift	SN = Signed numeric
Mod = Modified	NO = Numeric only	KT = Katakana
	OP = Open	EI = Either
	DN = DBCS Only	

Fld#	Start	End	Length	Unp	Nds	Int	Mod	Type	First 26 Columns
1	[1,2]	[2,24]	103					AN	[
2	[2,26]	[2,49]	24					AN	<DBCS FIELDS>]
3	[2,51]	[4,80]	190					AN	
4	[5,2]	[5,5]	4					AN	[OPEN]
5	[5,7]	[5,17]	11					AN	]
6	[5,19]	[5,34]	16	X		X		OP	[80980<98>]
7	[5,36]	[6,80]	125					AN	
8	[7,2]	[7,7]	6					AN	[EITHER]
9	[7,9]	[7,17]	9					AN	]
10	[7,19]	[7,34]	16	X		X		EI	<どちらかのみ >]
11	[7,36]	[8,80]	125					AN	
12	[9,2]	[9,9]	8					AN	[DBCSONLY]
13	[9,11]	[9,17]	7					AN	]
14	[9,19]	[9,34]	16	X		X		DN	<漢字のみ >]
15	[9,36]	[10,80]	125					AN	
16	[11,2]	[11,9]	8					AN	[KANAKANA]
17	[11,11]	[11,17]	7					AN	]
18	[11,19]	[11,34]	16	X		X		KT	[ldldlkf!ke!kf!ke]
19	[11,36]	[11,80]	45					AN	
20	[12,2]	[12,13]	12					AN	[ALPHANUMERIC]
21	[12,15]	[12,17]	3					AN	[]
22	[12,19]	[12,34]	16	X		X		AN	[LKA:;:;:;JDFKL]
23	[12,36]	[12,80]	45					AN	
24	[13,2]	[13,11]	10					AN	[ALPHA_ONLY]
25	[13,13]	[13,17]	5					AN	[]
26	[13,19]	[13,34]	16	X		X		AB	[LKJALLKJLKJFLKJL]
27	[13,36]	[13,80]	45					AN	
28	[14,2]	[14,14]	13					AN	[NUMERIC_SHIFT]
29	[14,16]	[14,17]	2					AN	[]
30	[14,19]	[14,34]	16	X		X		NS	[LKSDJ9999999666]
31	[14,36]	[14,80]	45					AN	
32	[15,2]	[15,13]	12					AN	[NUMERIC_ONLY]
33	[15,15]	[15,17]	3					AN	[]
34	[15,19]	[15,34]	16	X		X		NO	[9890898039840984]
35	[15,36]	[15,80]	45					AN	
36	[16,2]	[16,11]	10					AN	[DIGIT_ONLY]
37	[16,13]	[16,17]	5					AN	[]
38	[16,19]	[16,34]	16	X		X		DO	[0938403984094804]
39	[16,36]	[16,80]	45					AN	
40	[17,2]	[17,11]	10					AN	[MAG_STRIPE]
41	[17,13]	[17,17]	5					AN	[]
42	[17,19]	[17,34]	16	X				MS	]
43	[17,36]	[17,80]	45					AN	
44	[18,2]	[18,15]	14					AN	[SIGNED_NUMERIC]
45	[18,17]	[18,17]	1					AN	[]
46	[18,19]	[18,35]	17	X		X		SN	[09809]
47	[18,37]	[24,80]	524					AN	

1.18.3 M5250 Module Action Routines

1.18.3.1 action M52_Init and action M52_Connect

Actions **M52_Init** and **M52_Connect** have the following additional input variables in addition to the input variable **M52_SessionName**.

<i>Input</i>	M52_DeISOSI	(boolean)
<i>Variables:</i>	M52_SO	(string)
	M52_SI	(string)
	M52_CaseInsensitive	(boolean)

Remarks: The boolean variable **M52_DeISOSI**, when set to **true**, discards the SO and SI characters when reading host screen data into EASEL OS/2 EE objects. If set to **false**, the default value, the SO and SI characters remain and are translated to the string value of the input variables **M52_SO** and **M52_SI** respectively. The values of **M52_SO** and **M52_SI** must be assigned values which are 0 or 1 byte in length each. If they are not initialized, they are treated as SBCS blanks.

The **M52_DeISOSI** option affects only functions that read host screens such as **M52_ReadField**, **M52_ReadLine**, **M52_ReadScreen** and **M52_ReadColoredScreen**, not functions which write to host screens.

M52_CaseInsensitive, when set to **true**, executes read and watch actions as case-insensitive. The default value is **false**, case-sensitive.

Example: response to Start5250

```
copy "A" to M52_SessionName
copy "<" to M52_SO           # display SO as "<"
copy ">" to M52_SI           # display SI as ">"
copy true to M52_CaseInsensitive # case-insensitive
action M52_Init
```

The values of these variables are used in subsequent **M52_Connect** action routines. For example, if you issue action **M52_Connect** after the above example without specifying **M52_SessionName**, then **M52_Connect** tries to connect to Session A. The **M52_SO** and **M52_SI** variable values set by **M52_Init** are kept as the default values and may be used for all sessions. To use different variable values for a specific session, copy the new values to the variable, and then connect to that session. These new values then become the new defaults. To reset the values to their original values, the **copy** statement can be used anytime following **M52_Connect**. For example:

```
copy "B" to M52_SessionName      # connect session B
copy false to M52_CaseInsensitive # only for session B
action M52_Connect
copy true to M52_CaseInsensitive # reset to previous value
```

Note: If you omit the last line `copy true to M52_CaseInsensitive`, all **M52_Connect** actions thereafter use the value **false** for **M52_CaseInsensitive** unless a different value is specified explicitly.

1.18.3.2 action M52_GetFieldAttributes

M52_GetFieldAttributes has one additional output variable which contains the type of the field.

<i>Output Variables:</i>	FieldType	(integer)
<i>Remarks:</i>	GetFieldAttributes sets FieldType to one of the following constants:	
	M52_FT_KATAKANA	For Katakana fields
	M52_FT_DBCSONLY	For DBCS only fields
	M52_FT_OPEN	For DBCS and SBCS character fields
	M52_FT_EITHER	For DBCS only or SBCS only fields (decided by the first character entered)

The above constants are defined in the DBCS.INC include file.

1.18.3.3 Bisected DBCS Characters

In read actions **M52_ReadLine**, **M52_ReadScreen**, and **M52_ReadColoredScreen**, DBCS characters can be bisected at the borders of the EASEL OS/2 EE areas holding the host data. If this happens, EASEL OS/2 EE replaces the fragment bytes of bisected DBCS characters with SBCS blanks.

1.18.3.4 M52_Watch Action Routines

Watch actions **M52_WatchForChar** and **M52_WatchForNotChar** support watching for DBCS characters. For example the following watch definition is valid.

```
action Sequence_4 is
  action M52_DefineWatch
  copy 20 to M52_WatchRow
  copy 12 to M52_WatchCol
  copy "円" to M52_WatchChar
  action M52_WatchForChar
```

When a DBCS character is being watched, the variable **M52_WatchCol** signifies the first byte of the DBCS character.

1.18.3.5 action **M52_FindLastNonBlankLine**

M52_FindLastNonBlankLine returns the number of the last non-blank line on the screen preceding the line on which the cursor lies. With this action you can skip blank lines to find, for example, the most recent prompt or message. **M52_FindLastNonBlankline** regards DBCS blanks as blanks, as well.

1.18.3.6 action **M52_WriteField**

M52_WriteField supports mixed strings for the input variable **M52_FieldText**. **M52_FieldText** can be longer than the specified length of **M52_FieldNumber** as a result of the insertion of SO and SI characters. In this case, trailing characters are truncated and an error is returned to the application. Errors are returned only when truncation occurs due to insertion of SO and SI characters.

1.18.3.7 action **M52_PressKey**

To support DBCS, there is an additional key code for **M52_PressKey**.

M52_VERT_BROKEN_LINE For a vertical broken line

1.18.3.8 action **M52_PrintScreen**

M52_PrintScreen has an input variable to specify how to print SO and SI characters.

Input Variables: **M52_PrintSOSI** (boolean)

Remarks: If **M52_PrintSOSI** is set to **true**, SO and SI characters are printed as “<” and “>” respectively. If set to **false**, the default value, SO and SI characters are printed as SBCS blanks.

1.19 3270/5250 Support

3270/5250 Support does not provide DBCS support.

1.20 Printing and Plotting

1.20.1 plot Action Statement

The **plot** action statement prints or plots graphical and textual regions using outline fonts. For this reason, it is necessary to install the optional *OS/2 DBCS Outline Fonts* package in order for the **plot** action statement to work correctly.

1.20.2 Printing Text Files

The ESLPRS.EXE and ESLPR.EXE utilities support DBCS file names and path names. In the example below you can specify a DBCS path name and file name for FILENAME.

```
application PrintFile
:
response to start
    start local PrintFile "eslprs.exe"
:
response to PrintKey
    send "file FILENAME pagesize 40¥n" to PrintFile
```

Chapter 2. 3270/5250 Support

2.1 3270/5250 Support Differences

2.1.1 General Information

3270/5250 Support for EASEL OS/2 EE is a combination of 3270 Support and 5250 Support. While most of the functionality of both of these features has been retained, they now share the same subroutines. These subroutines are contained in a stimulus Dynamic Link Library (DLL), rather than an action module. The DLL is called the EASEL OS/2 EE Communications Server (ECS), and provides directly callable subroutines to the EASEL OS/2 EE programmer.

Previously, all communications with a local host were done through global variables. With 3270/5250 Support, communications take place through user-defined parameters.

All subroutines are in the form **Ecs**[subroutine-name].

The following table lists the differences between the 3270 and 5250 Support action routines and the DLL subroutines.

- Some subroutines have been deleted and have not been replaced with subroutines.
- Other routines have been combined into a single subroutine.
- Some subroutines are completely new.

Subsection	Old Action Routine(s)	New Subroutine(s)
Initialization and Termination	Init and Stop	Now called transparently.
Connecting and Disconnecting Sessions	Connect, Disconnect, M52_Connect, and M52_Disconnect	EcsConnect EcsDisconnect
Examining the Screen Buffer	ScanScreen and M52_ScanScreen	Now done automatically.
		EcsGetFieldCount
	GetFieldAttributes and M52_GetFieldAttributes	EcsGetFieldType EcsGetFieldAttribute
	FindLastNonBlankLine and M52_FindLastNonBlankLine	EcsGetLastNonBlankLine-Number

Subsection	Old Action Routine(s)	New Subroutine(s)
Synchronization	M52_InformWhenIIOff and M52_WaitForIIOff	Deleted. No new subroutine.
	DefineWatch and M52_DefineWatch	EcsClearWatch
	WatchForChar and M52_WatchForChar	EcsWatchForStringAt
	WatchForNotChar and M52_WatchForNotChar	EcsWatchForStringNotAt
	Watch and WatchAndWait	EcsWatch and EcsWatchAndWait. Functionality has changed.
	WatchForNoX	Deleted. EcsWatchForIIOff offers identical functionality.
		EcsWatchForScreenSize
Communicating with the Host		EcsSetAutoWait
	PressKey, PressPFKey, M52_PressKey, and M52_PressCMDKey	EcsPressKey
Controlling the Emulator Window	SetEmulatorWindow and Emulate3270	EcsSetEmulatorWindow, EcsEmulate, and EcsEmulateAndWait
Miscellaneous	Print3270Screen and M52_PrintScreen	Deleted. No new subroutines.
		EcsGetError
	GetXStatus M52_CheckIndicators	EcsCheckIndicator
		EcsGetCurrentSession
	FIELD52.EXE	ECSFIELD.EXE

The following sections discuss these changes in greater detail.

2.1.2 New Subroutines

2.1.2.1 Connecting and Disconnecting Sessions

3270/5250 Support for EASEL OS/2 EE provides two new subroutines, **EcsConnect** and **EcsDisconnect**, to connect and disconnect EASEL OS/2 EE programs to and from sessions.

2.1.2.2 Examining the Screen Buffer

A new EASEL OS/2 EE subroutine, **EcsGetFieldCount**, enables you to find out how many fields the current screen has. This function used to be performed by **ScanScreen** and **M52_ScanScreen**, which have been deleted.

The 3270 action routine **GetFieldAttributes** has been replaced with the subroutines **EcsGetFieldType** and **EcsGetFieldAttribute**.

- To retrieve a field's type, you now call **EcsGetFieldType** specifying the field number. On return, FieldType holds an integer that specifies the field's type.
- To retrieve a field's attributes, you now call **EcsGetFieldAttribute**, specifying the field by number and the desired attribute.

FindLastNonBlankLine and **M52_FindLastNonBlankLine** have been replaced with **EcsGetNonBlankLineNumber**.

2.1.2.3 Synchronization

DefineWatch and **M52_DefineWatch** are now **EcsClearWatch**.

WatchForChar and **M52_WatchForNotChar** are now **EcsWatchForStringAt**.

WatchForNotChar and **M52_WatchForNotChar** are now **EcsWatchForStringNotAt**.

EcsWatchForScreenSize watches for the number of rows and columns of the next screen, usually associated with changing 3270 Emulator screen size among different screens.

Refer to 2.1.5, "Watch Responses" on page 2-7, for more information on watches.

2.1.2.4 Communicating with the Host

Instead of passing **EcsAutoWaitFlag** to all communications routines as a parameter, since it is a global variable, you can now set it with a call to **EcsSetAutoWait**.

FillSwitch specifies how a particular field will be filled with data. This is more useful with 3270 sessions, since a BLANK is the only legal fill character for a 5250 session.

2.1.2.5 Controlling the Emulator Window

3270/5250 Support provides three subroutines for controlling the 3270 emulator. These subroutines are:

EcsSetEmulatorWindow
EcsEmulate
EcsEmulateAndWatch

EcsSetEmulatorWindow enables you to change the state of the 3270 emulator window associated with the current session.

EcsEmulate places you in the 3270 windowed emulator by making it visible, active, and maximized.

EcsEmulateAndWatch is similar to **EcsEmulate** in that it places you in the 3270 windowed emulator by making it visible, active, and maximized. However, it also enables you to specify one or more screens that, when encountered by the emulator, cause the action to return control to the EASEL OS/2 EE application automatically.

Note: These commands are available for 3270 only.

2.1.2.6 Miscellaneous Commands

The error-handling capabilities of 3270/5250 Support have been overhauled substantially. Now, the built-in **errorlevel** is set during each ECS call except:

EcsSetAutoWait
EcsClearWatch
EcsWatchForCursorAt
EcsWatchForCursorNotAt
EcsWatchForScreenSize
EcsWatchForStringAt
EcsWatchForStringNotAt
EcsWatchForIIOff

These calls are internal subroutines that manipulate EASEL OS/2 EE variables and thus actually do not make any calls to the DLL.

In addition, the EASEL OS/2 EE internal subroutine **ErrorHandler** has been included for use by the EASEL OS/2 EE internal subroutines that are defined in the ESLCMSRV.INC file. This subroutine also serves as a guide to the intended procedure concerning error handling. After each ECS call, except for those listed above, the value of the built-in **errorlevel** should be checked. If an error is detected, the value of **errorlevel** should be saved, as it is extremely volatile.

For instance, every call to the DLL and every send action affects the value of **errorlevel**. Once the **errorlevel** is in an EASEL OS/2 EE variable, **EcsGetError** can be called to retrieve the text that is associated with that error.

GetXStatus and **M52_CheckIndicators** have been combined into **EcsCheckIndicator**. Now, you can call **EcsCheckIndicator** and specify which indicator you are interested in. On return, **IndicatorState** holds an integer that specifies the indicator's state.

2.1.3 Deleted Routines

This section lists the 3270 and 5250 action routines that have been deleted and have not been replaced with DLL subroutines.

2.1.3.1 Initialization and Termination

All initialization and termination routines for both 3270 (**Init3270** and **Stop3270**) and 5250 (**M52_Init** and **M52_Stop**) support have been deleted. Since the interface is now a stimulus DLL, Init and Stop are called transparently by EASEL OS/2 EE.

2.1.3.2 Examining the Screen Buffer

Routines **ScanScreen** and **M52_ScanScreen** have been deleted since they are now done automatically. Refer to 2.1.2.2, "Examining the Screen Buffer" on page 2-3 for more information.

GetFieldAttributes has been deleted and replaced by **EcsGetFieldType** and **EcsGetFieldAttribute**. Refer to 2.1.2.2, "Examining the Screen Buffer" on page 2-3 for more information.

FindLastNonBlankLine and **M52_FindLastNonBlankLine** are now **EcsGetLastNonBlankLineNumber**.

2.1.3.3 Synchronization

Routines **M52_InformWhenIOff** and **M52_WaitForIOff** have been deleted.

WatchForNoX (3270 Support) has been deleted. **EcsWatchForIOff** offers identical functionality.

2.1.3.4 Communicating with the Host

PressPFKey and **M52_PressCMDKey** have been deleted. With 3270/5250 Support, you can now have a function key specified for **EcsPressKey**.

2.1.3.5 Miscellaneous Commands

Print3270Screen and **M52_PrintScreen** have been deleted.

FIELDS52.EXE has been deleted. **ECSFIELD.EXE** provides the identical functionality.

2.1.4 Global Variables

Global variables are necessary to enable and facilitate the auto wait feature. The global variables used by the ECS internal subroutines include:

global boolean variable <code>EcsAutoWaitFlag</code>	is false
global boolean variable <code>EcsWaitGaveUp</code>	is false
global integer variable <code>EcsTimeLimit</code>	is 3000
global string variable <code>EcsWatchCommandString</code>	is ""

The **EcsAutoWaitFlag** is set by a call to **EcsSetAutoWait** with the integer parameter set to either **ECS_AUTOWAITOFF** or **ECS_AUTOWAITON**, indicating the state you want **EcsAutoWaitFlag** to have. If **EcsAutoWaitFlag** has been set **ON**, whenever you call **EcsPressKey**, **EcsEnterString**, or **EcsPressENTER**, a call to **EcsWatchAndWait** is made.

The parameters used by the AutoWait facility are the global variables **EcsTimeLimit** and **EcsWatchCommandString**. The default time limit is three seconds. The default watch command string is a null string, which is not a legal watch command string. It is important to set this string explicitly. Setting the auto wait feature to **ON** and calling one of the three subroutines listed above (**EcsPressKey**, **EcsEnterString**, or **EcsPressENTER**) will, therefore, cause an error unless the developer has set the global variable

EcsWatchCommandString to the desired wait criteria.

You should check the global variable **EcsWaitGaveUp** after any of these three calls is made, to determine the outcome of the **EcsWatchAndWait** call that is made automatically.

2.1.5 Watch Responses

3270/5250 Support supports multiple watch command strings, each of which can be cleared and set individually. In addition, because the watch command string is now a parameter, you can have as many watch command strings as you want. You can, however, have only one active watch at a time.

The timing of 3270/5250 Support has been changed from hundredths of a second to thousandths of a second (from .01 to .001). For example, now a time limit of 3000 means 3 seconds, not 30.

In addition, the responses that are necessary for watches have changed. Since ECS is a stimulus DLL, the responses must be in the following form:

```
response to stimulus EaselComServer
[EventFilter] [ParameterFilter]
```

The particular event filters are defined as follows:

```
integer constant
    ECS_WATCHDONE           is 1
    ECS_WATCHGAVEUP        is 2
    ECS_WATCHERROR         is 3
    ECS_EMULATEWATCHDONE   is 4
    ECS_EMULATEWATCHERROR  is 5
```

If the event filter is an error, there is also a parameter filter that is the error number. You should examine the implementation of **EcsWatchAndWait** in the ESLCMSRV.INC file as a guide to the intended procedure concerning setting up a block of responses for the **EcsWatch** call.

Also, after every **EcsWatch** call, it is necessary to reconnect to the session. Refer to the ESLCMSRV.INC file for an example.

2.1.6 The Include File

The 3270/5250 Support include file (ESLCMSRV.INC) is an essential part of the EASEL OS/2 EE Communications Server. All parameters must be defined by the user. The include file contains:

- The declaration of the stimulus.
- The declarations of the ECS external routines.
- The definitions of the internal subroutines.
- All constants used.
- The global variables used by the ECS.

The include file provides the following external subroutines for use by the developer:

```
EcsConnect ( SessionShortName )
EcsDisconnect ( )
EcsGetCurrentSession ( SessionShortName )
EcsReadField ( FieldNumber, FieldContents )
EcsReadLine ( Row, LineContents )
EcsGetFieldCount ( FieldCount )
EcsGetFieldNumber ( Row, Column, FieldNumber )
EcsGetFieldPosition ( FieldNumber, Row, Column )
EcsGetFieldLength ( FieldNumber, FieldLength )
EcsGetFieldType ( FieldNumber, FieldType )
EcsGetFieldAttribute ( FieldNumber,
    FieldAttribute, AttributeState )
EcsGetCursorPosition ( Row, Column )
EcsGetLastNonBlankLineNumber ( StartLine,
    LineNumber )
EcsWatch ( TimeLimit, WatchCommandString )
EcsTypeString ( String )
EcsWriteField ( FieldNumber, String, FillSwitch )
EcsCheckIndicator ( Indicator, IndicatorState )
```

```

EcsGetSessionStatus ( SessionStatusIndicator)
EcsGetSessionInfo ( SessionShortName, Type, Rows,
    Columns )
EcsGetError ( ErrorLevel, ErrorMessage )

```

The following internal subroutines are provided for use by the developer:

```

EcsReadScreen ( Top, Left, Bottom, Right,
    TextRegion)
EcsReadColoredScreen ( Top, Left, Bottom, Right,
    ColoredTextRegion )
EcsClearWatch ( WatchCommandString )
EcsWatchForCursorAt ( Row, Column,
    WatchCommandString)
EcsWatchForCursorNotAt ( Row, Column,
    WatchCommandString )
EcsWatchForScreenSize ( Rows, Columns,
    WatchCommandString )
EcsWatchForStringAt ( Row, Column, String,
    WatchCommandString )
EcsWatchForStringNotAt ( Row, Column, String,
    WatchCommandString )
EcsWatchForIIOff ( SettleTime, WatchCommandString )
EcsWatchAndWait ( TimeLimit, WatchCommandString )
EcsSetAutoWait ( AutoWaitFlag )
EcsPressKey ( KeyCode )
EcsPressENTER ( )
EcsEnterString ( String )
EcsSetEmulatorWindow ( Command )
EcsEmulate ( )
EcsEmulateAndWatch ( ScreenIDFile, ScreenName )

```

The following external subroutines are declared, but not intended for use by the developer:

```

EcsGetScreen ( Top, Left, Bottom, Right,
    ScreenContents )
EcsGetColoredScreen ( Top, Left, Bottom, Right,
    ColoredTextRegion )
EcsSendKey ( KeyCode )
EcsDoEmulatorCommand ( Command )
EcsDoEmulate ( )
EcsDoEmulateAndWatch ( ScreenIDFile, ScreenName )
EcsSetError ( ErrorNumoer )

```

The following internal subroutines are defined but not intended for use by the developer:

```

ErrorHandler ( NoError, EcsErrorLevel )

```

2.2 Introduction

2.2.1 EASEL OS/2 EE in the 3270/5250 Environment

3270 and 5250 are the generic names of a family of IBM terminals, controllers, and associated software. 3270/5250 applications transmit information on a screen-by-screen basis.

With the 3270/5250 Support for EASEL OS/2 EE, a full screen of information is held in a screen buffer, where its data can be accessed. The 3270/5250 Support for EASEL OS/2 EE provides facilities that allow you to examine data in the buffer, place data into specified fields, transfer data to the host, and execute other functions that can be performed by a 3270/5250 terminal. These functions are accomplished by calling the EASEL OS/2 EE Communications Server (ECS) subroutines described in this chapter.

The ECS is a stimulus dynamic link library (DLL) interface that provides directly callable subroutines to the EASEL OS/2 EE programmer. A single set of subroutines, including some EASEL OS/2 EE subroutines, supports both 3270 and 5250 sessions. For any given environment, a single DLL and its include file (containing function prototypes, stimulus declaration, various constant definitions, and several EASEL OS/2 EE subroutines) are provided.

2.2.2 Components of 3270/5250 Support for EASEL OS/2 EE

The following 3270/5250 Support components allow you to create an EASEL OS/2 EE interface to a 3270/5250 application.

2.2.2.1 ECS Include File

The ECS include file defines EASEL OS/2 EE internal subroutines and declares EASEL OS/2 EE external subroutines that are specific to the 3270/5250 environment. It also defines EASEL OS/2 EE variables and constants used by the ECS subroutines. These subroutines

- Read the contents of the screen buffer into an EASEL OS/2 EE textual region.
- Read a line into a string variable.
- Read a field's contents into a string variable.
- Write the contents of a string variable to a field.
- Send information to the host.
- Simulate the pressing of a 3270/5250 terminal's special function keys.

2.2.2.2 ECSFIELD.EXE EASEL OS/2 EE Development Tool

The ECSFIELD.EXE EASEL OS/2 EE development tool provides you with essential information about each field on the current screen in the 3270/5250 screen buffer.

2.2.3 Using the ECS: Required Procedures

The following steps are required to successfully implement an EASEL OS/2 EE program that uses the ECS.

1. Install IBM OS/2 Extended Edition Communications Manager (CM), if it is not already installed. Configure CM for one or more 3270/5250 terminal emulation sessions.
2. Make sure 3270/5250 terminal emulation is started before running an EASEL OS/2 EE program that uses the ECS, or before running the ECSFIELD.EXE utility. You can start 3270/5250 terminal emulation from the CM "Start Communications" menu, or you can configure the CM to start 3270/5250 communications automatically when the CM is started. Refer to the Using Communications Manager section in your OS/2 Extended Edition manual for more information.
3. Where necessary, examine the output generated by ECSFIELD.EXE to identify the precise contents of the buffer.
4. Write a test program that
 - a. References the ECS.
 - b. Initializes 3270/5250 communications and connects to one or more sessions.
 - c. Synchronizes the timing of 3270/5250 and EASEL OS/2 EE via ECS statements, so that the screen buffer is filled before subroutine calls are made to retrieve the information.
 - d. Reads specific buffer contents into EASEL OS/2 EE string variables and textual regions, and communicates with the host via ECS subroutines.
5. Write the final program, incorporating step 4 with other EASEL OS/2 EE subroutine and response statements.

2.2.4 Declaring the ECS Stimulus Library

To declare the ECS stimulus library in an EASEL OS/2 EE program, include the following statement at the very beginning of the program:

```
include eslcmshr.inc
```

2.2.5 Sample Program

The following example shows a complete EASEL OS/2 EE program for a 3270/5250 application that contains one full screen buffer.

```
# This include file contains definitions of all ECS subroutines,
# constants, and a stimulus declaration.

include "eslcmrv.inc"

# Defines screen size and default font.

screen size device units
font "medium"

#
# Variable Declarations
#

# Variables will contain top, left, bottom, and right margins of a
# screen buffer, and current error we are working with.

integer variable Top
                Left
                Bottom
                Right
                ErrorLevel

# Variables will contain session name, name of textual region, and
# error message.

string variable SessionShortName
                TextRegion
                Message

#
# Object Declarations
#

enabled visible color 26 primary graphical region First
    size 636 454
    at position 2 3
    window size 636 454 at position 0 0
    black foreground
    title bar "MAIN SCREEN"
    system menu
    no scale
    minimize button
    maximize button
```

```
enabled visible white textual region Screen
  window size 80 columns 25 lines
  at position 0 95 in First
  black foreground
  title bar "EMULATOR SCREEN"
  system menu
```

```
enabled invisible white textual region Errors
  window size 53 columns 3 lines
  at position 95 165 in First
  black foreground
  size border
  title bar " ERRORS "
```

response to start

```
# Initialize SessionShortName with desired session name.

copy "F" to SessionShortName

# Clear textual region Screen, initialize margins of a screen
# buffer, and initialize TextualRegion variable with a name of
# textual screen which will display screen buffer information.

clear Screen
copy 1 to Top
copy 1 to Left
copy 24 to Bottom
copy 80 to Right
copy "Screen" to TextRegion

# If there are no errors we will read contents of a screen buffer,
# else we will process the current error with a call to DLL.
# Textual region Errors will be updated and will contain current
# error level and corresponding message.

call EcsConnect( SessionShortName )
if ( errorLevel = 0 ) then
  call EcsReadScreen( Top, Left, Bottom, Right, TextRegion )
else
  copy errorLevel to ErrorLevel
  if ( ErrorLevel > 0 ) then
    clear Errors
    make Errors visible
    call EcsGetError( ErrorLevel, Message )
```

```
        add to Errors
            move to column 1 line 1
            insert "Error: " ErrorLevel
            move to column 1 line 3
            insert "ErrorText: " Message
        wait 5
        make Errors invisible
        send " Error: " ErrorLevel "\n" to errorlog
    end if
end if
```

response to Screen on close

```
# In response to Close from system menu, we will disconnect from
# current session and exit EASEL OS/2 EE program.

call EcsDisconnect()
exit
```

2.3 3270/5250 Session and Screen Concepts

2.3.1 Session Names

CM enables you to create multiple logical 3270/5250 terminal sessions. ECS subroutines allow EASEL OS/2 EE to control one or more of these sessions. Each session configured by CM has both a long session name and a short session name.

The long session name can be up to eight characters long. The short session name is a single letter, such as "A" or "D." ECS subroutines use only short session names for denoting sessions.

2.3.2 Session States

From the point of view of EASEL OS/2 EE, each configured session, once it has been started by CM, can be in one of three states:

- Disconnected
- Connected, not active
- Connected and active

2.3.2.1 Disconnected

When disconnected, the session is not controlled by EASEL OS/2 EE. It may, however, currently be connected by another process.

2.3.2.2 Connected, Not Active

When connected but not active, the session is controlled by EASEL OS/2 EE. However, it is not the session that is acted upon by the set of ECS subroutines that read, write, or send data. Other running processes are prevented from connecting to this session.

2.3.2.3 Connected and Active

When connected and active, the session is controlled by EASEL OS/2 EE and is acted upon by the set of ECS subroutines that read, write, or send data. Other running processes are prevented from connecting to this session.

At any given time, EASEL OS/2 EE can be connected to none, some, or all of the started 3270/5250 host sessions. If it is connected to one or more sessions, only one of these can be the active connected session; the rest, if any, are not active but remain connected. If there is no active connected

session, an error condition will result from calling any ECS subroutine that reads, writes, or sends data, returns status information, and so on.

The ECS subroutines that connect to or disconnect from 3270/5250 host sessions are:

EcsConnect
EcsDisconnect

EcsWatch disconnects from current session and user must reconnect by using **EcsConnect** call to DLL. (See 2.5.5.8, “EcsWatch” on page 2-82.)

2.3.3 Field-Oriented and Line-Oriented Screens

The 5250 screen buffer contains 24 lines by 80 columns and is generally field-oriented (or formatted), or organized by fields.

The 3270 screen buffer can have different display models:

MOD Number	Screen Size
Mod 2	24 x 80
Mod 3	32 x 80
Mod 4	43 x 80
Mod 5	27 x 132

Most of the ECS subroutines are written to access field-oriented screens. 3270/5250 Support for EASEL OS/2 EE accesses the screen buffer by first assigning sequential numbers to each field in the buffer. By specifying a field number, you can retrieve that field's contents and associated attributes, or write text into the specified field.

Sometimes, a 3270/5250 application contains screens wherein each line is a field of its own, or contains screens that have no fields at all. These are called line-oriented (or unformatted) screens.

There are two ECS subroutines used to access line-oriented screens:

EcsReadLine
EcsGetLastNonBlankLineNumber

You can use these subroutines to access line-oriented or field-oriented screens, but you cannot use the field-oriented subroutines to access line-oriented screens.

2.3.4 Field-Oriented (Formatted) Screens

By definition, a field is at least one character in length. A field can wrap around to the next line, or even to the beginning of the screen buffer. Each field is preceded by a field attribute character that defines the location and attributes of the field, but is not part of the field itself. Each field extends to

the next field attribute character that defines the next field in the screen. The following attributes are defined by the field attribute character:

Attribute	Description
protected/unprotected	Indicates whether the field is protected. A protected field cannot be modified (written to) by the user.
modified/unmodified	Indicates whether the field was modified (written).
display/nondisplay	Indicates whether the field is displayed on a 3270/5250 terminal.
intensified/normal	Indicates the brightness of the characters in the field.

The field attribute character also indicates one of the following field types:

Field Type	Description
Alphanumeric *	All characters are allowed.
Alphabetic only	Uppercase and lowercase letters, comma, period, hyphen, blank, and Dup key are allowed.
Numeric shift	Automatic shift for numerics is enabled.
Numeric only *	Digits 0-9, comma, period, plus, minus, blank, and Dup key are allowed.

*** Note:** These are the only valid field types for a 3270 session.

Field Type	Description
Digits only	Digits 0-9 and Dup key are allowed.
Magnetic stripe reader	Data can be entered only via a magnetic stripe reader.
Signed numeric	Digits 0-9, plus, minus, and Dup key are allowed.

Also, the 3270/5250 application assigns colors to fields, along with any, or none, of these visual attributes:

- reverse video
- underscore
- blink
- column separators

Of these, EASEL OS/2 EE supports the color attribute, so that you can examine characters for foreground and background colors once they have been read into a colored textual region.

Character positions within the 3270/5250 buffer are addressed by row and column. Row numbers go from 1 to 24, from top to bottom; column numbers go from 1 to 80, from left to right. The following examples illustrate how a character position is expressed as an ordered pair of numbers for Mod 2 of 3270 Emulator Screen:

[1,2] = row 1, column 2

[24,80] = row 24, column 80

Sometimes a host message is temporarily displayed on the 25th row, which is normally where the Operator Information Area is displayed. In this situation, row numbers go from 1 to 25. (See 2.5.4.9, "EcsReadLine" on page 2-56.)

Note: This is true for 5250 only.

Characters within the 3270/5250 Operator Information Area (the bottom line of the host system when there is no message being displayed) cannot be accessed directly via 3270/5250 Support for EASEL OS/2 EE. (See 2.5.8.1, "EcsCheckIndicator" on page 2-111 and, for 3270 sessions, see 2.5.8.2, "EcsGetSessionStatus" on page 2-113.)

2.4 Retrieving Field Information with ECSFIELD.EXE

ECSFIELD.EXE is a development tool that generates a report containing up to three sections:

- Information about the current screen
- Field summary
- Full field contents

2.4.1 Reading the ECSFIELD.EXE Report

2.4.1.1 Information About the Current Screen

The information about the current screen includes the following information:

- Index by column number
- Entire screen contents
- Number of fields in the screen
- Current cursor position, by row and column

2.4.1.2 Field Summary

The field summary includes the following information for each field in the screen:

- Field number
- Starting and ending field positions, by row and column
- Field length
- Field attributes
- First 26 characters in the field

2.4.1.3 Full Field Contents

The full field contents lists all characters in each field.

By default, only the first two sections are reported. In general, the field summary section should provide all the field information you need to develop an EASEL OS/2 EE application. (See 2.4.3, "ECSFIELD.EXE Sample Output" on page 2-21.)

2.4.2 Using ECSFIELD.EXE

To use ECSFIELD.EXE:

1. Access the 3270/5250 host application (using the CM 3270/5250 emulator) and go to the host screen for which you need the field information.

2. Activate an OS/2 fullscreen or windowed command prompt.
3. At the OS/2 command prompt, invoke ECSFIELD.EXE as follows:

Model

```
ECSFIELD [OPTIONS] [OUTPUT_FILENAME]
```

Where OPTIONS is a string containing any of the following parameters:

Parameter	Meaning
-sNAME	Report is for the 3270/5250 terminal session represented by the specified single-letter session short name. This parameter defaults to the first 3270/5250 host session listed when you run ECSFIELD.EXE with the “-L” option. It is helpful to specify this parameter, if user is interested in particular session’s screen information, when CM is configured for more than one 3270/5250 terminal emulation session.
-b	Report contains screen information section only.
-f	Report contains all three sections.
-L	Report lists all 5250 and 3270 host sessions started by CM. No field information is given. For each session, its short name, long name, screen dimensions, and type are shown. Sessions that have not been started are not listed.

Note: These parameters can be either upper- or lowercase.

OUTPUT_FILENAME is the name and extension of the OS/2 file in which the FIELDS report is placed. If you omit OUTPUT_FILENAME, the report is listed on your workstation’s display, and the screen information section does not show the screen contents. **Note:** If “-L” is specified, this parameter is ignored; the sessions are always listed on your workstation’s display.

2.4.2.1 Examples:

```
ECSFIELD
ECSFIELD login.fl d
ECSFIELD -b > news4.fl d
ECSFIELD -f -sB > product.fl d
```

2.4.3 ECSFIELD.EXE Sample Output

The following is sample output for the **-L** parameter:

ECSFIELD utility - Version H2.0 (EHLAPI)
International Business Machines Corporation
(C) Copyright Easel Corporation 1990
All Rights Reserved.

Number of started host sessions = 10

Short Name	Long Name	Rows	Columns	Type
-----	-----	----	-----	-----
A	A	24	80	3270 Host
B	B	32	80	3270 Host
C	C	43	80	3270 Host
D	D	27	132	3270 Host
E	E	0	0	3270 Printer
F	F	24	80	5250 Host
G	G	24	80	5250 Host
H	H	24	80	5250 Host
I	I	24	80	5250 Host
J	J	0	0	5250 Printer

The following page shows a sample output from ECSFIELD.EXE when no options are specified.

ECSFIELD utility - Version H2.0 (EHLLAPI)
International Business Machines Corporation
(C) Copyright Easel Corporation 1990
All Rights Reserved.

Session short name = A

Screen contains 49 fields.
Cursor is at [20,16].

Field summary:

Unp = Unprotected	AN = Alphanumeric	DO = Digits only
Nds = Nondisplay	AB = Alphabetic only	MS = Mag stripe data only
Int = Intensified	NS = Numeric shift	SN = Signed numeric
Mod = Modified	NO = Numeric only	

Fld#	Start	End	Length	Unp	Nds	Int	Mod	Type	First 26 Characters
1	[1,2]	[1,51]	50			X		AN	[Virtual Machine/System Pro
2	[1,53]	[17,80]	4					AN	[
3	[18,2]	[18,80]	79			X		NO	[Fill in your USERID and PA
4	[19,2]	[19,80]	79			X		NO	[(Your Password will not ap
5	[20,2]	[20,14]	13			X		NO	[USERID ==>]
6	[20,16]	[20,23]	8	X			X	AN	[
7	[20,25]	[20,80]	56					NO	[
8	[21,2]	[21,14]	13			X		NO	[PASSWORD ==>]
9	[21,16]	[21,55]	40	X	X		X	AN	[
10	[21,57]	[22,80]	104					NO	[
11	[23,2]	[23,14]	13			X		NO	[COMMAND ==>]
12	[23,16]	[24,59]	124	X			X	AN	[
13	[24,61]	[24,79]	19					AN	[RUNNING HOME]

The following page shows a sample output from ECSFIELD.EXE when option **-b** is specified.

ECSFIELD utility - Version H2.0 (EHLLAPI)
International Business Machines Corporation
(C) Copyright Easel Corporation 1990
All Rights Reserved.

Session short name = A

Screen contains 49 fields.
Cursor is at [23,6]

The following page shows a sample output from ECSFIELD.EXE when option
-f is specified.

Session short name = A

Screen contains 49 fields.
Cursor is at [23,6].

Field summary:

Unp = Unprotected	AN = Alphanumeric	DO = Digits only
Nds = Nondisplay	AB = Alphabetic only	MS = Mag stripe data only
Int = Intensified	NS = Numeric shift	SN = Signed numeric
Mod = Modified	NO = Numeric only	

Fld#	Start	End	Length	Unp	Nds	Int	Mod	Type	First 26 Characters
1	[1,2]	[1,76]	75			X		AN	[
2	[1,78]	[2,1]	4			X		AN	[A00]
3	[2,3]	[3,1]	79					AN	[Press one of the following
4	[3,3]	[3,6]	4			X		AN	[PF1]
5	[3,8]	[3,55]	48					AN	[Process calendars
6	[3,57]	[4,1]	25					AN	[Time: 2:49 PM]
7	[4,3]	[4,6]	4			X		AN	[PF2]
8	[4,8]	[5,1]	74					AN	[Open the mail
9	[5,3]	[5,6]	4			X		AN	[PF3]
10	[5,8]	[5,48]	41					AN	[Find documents
11	[5,50]	[6,1]	32					AN	[1990 AUGUST 1990
12	[6,3]	[6,6]	4			X		AN	[PF4]
13	[6,8]	[6,48]	41					AN	[Process notes and messages
14	[6,50]	[7,1]	32					AN	[S M T W T F
15	[7,3]	[7,6]	4			X		AN	[PF5]
16	[7,8]	[7,48]	41					AN	[Prepare documents
17	[7,50]	[8,1]	32					AN	[1 2 3
18	[8,3]	[8,6]	4			X		AN	[PF6]
19	[8,8]	[8,48]	41					AN	[Process documents from oth
20	[8,50]	[8,58]	9					AN	[5 6]
21	[8,60]	[8,60]	1			X		AN	[7]
22	[8,62]	[9,1]	20					AN	[8 9 10 11]
23	[9,3]	[9,6]	4			X		AN	[PF7]
24	[9,8]	[9,48]	41					AN	[Process the mail log
25	[9,50]	[10,1]	32					AN	[12 13 14 15 16 17 1
26	[10,3]	[10,6]	4			X		AN	[PF8]
27	[10,8]	[10,48]	41					AN	[Check the outgoing mail
28	[10,50]	[10,80]	31					AN	[19 20 21 22 23 24 2
29	[11,2]	[11,48]	47			X		AN	[
30	[11,50]	[11,80]	31					AN	[26 27 28 29 30 31
31	[12,2]	[12,6]	5			X		AN	[PF10]
32	[12,8]	[12,48]	41					AN	[Add an automatic reminder
33	[12,50]	[12,80]	31					AN	[Day of Year: 21
34	[13,2]	[13,6]	5			X		AN	[PF11]
35	[13,8]	[14,6]	79					AN	[View main menu number 2
36	[14,8]	[14,50]	43					AN	[5664-309 (C) Copyright IBM
37	[14,52]	[14,59]	8			X		AN	[PF9]
38	[14,61]	[14,80]	20					AN	[Help PF12 End]
39	[15,2]	[15,79]	78			X		AN	[-----
40	[16,1]	[16,79]	79					AN	[
41	[17,1]	[17,79]	79					AN	[

42		[18,1]	[18,79]	79						AN		[
43		[19,1]	[19,79]	79						AN		[
44		[20,1]	[20,79]	79						AN		[
45		[21,1]	[21,79]	79						AN		[
46		[22,1]	[22,79]	79						AN		[
47		[23,1]	[23,4]	4				X		AN		[==>]
48		[23,6]	[24,45]	120		X		X		AN		[
49		[24,47]	[24,80]	34				X		AN		[

Full field contents:

```

*** Field 1:
[                                PROFS MAIN MENU                                ]

*** Field 2:
[A00 ]

*** Field 3:
[Press one of the following PF keys.                                ]

*** Field 4:
[PF1 ]

*** Field 5:
[Process calendars                                ]

*** Field 6:
[Time:   2:49 PM                                ]

*** Field 7:
[PF2 ]

*** Field 8:
[Open the mail                                ]

*** Field 9:
[PF3 ]

*** Field 10:
[Find documents                                ]

...*** Field 11:
[ 1990   AUGUST   1990                                ]

*** Field 12:
[PF4 ]

*** Field 13:
[Process notes and messages                                ]

*** Field 14:
[ S  M  T  W  T  F  S                                ]

*** Field 15:
[PF5 ]

*** Field 16:
[Prepare documents                                ]

*** Field 17:
[      1      2      3      4                                ]

```

```

*** Field 18:
[PF6 ]

*** Field 19:
[Process documents from other sources ]

*** Field 20:
[ 5 6 ]

*** Field 21:
[7]

*** Field 22:
[ 8 9 10 11 ]

*** Field 23:
[PF7 ]

*** Field 24:
[Process the mail log ]

*** Field 25:
[ 12 13 14 15 16 17 18 ]

*** Field 26:
[PF8 ]

*** Field 27:
[Check the outgoing mail ]

...*** Field 28:
[ 19 20 21 22 23 24 25 ]

*** Field 29:
[ ]

*** Field 30:
[ 26 27 28 29 30 31 ]

*** Field 31:
[PF10 ]

*** Field 32:
[Add an automatic reminder ]

*** Field 33:
[ Day of Year: 219 ]

*** Field 34:
[PF11 ]

*** Field 35:
[View main menu number 2 ]

*** Field 36:
[5664-309 (C) Copyright IBM Corp. 1983, 1988]

*** Field 37:
[ PF9]

*** Field 38:
[Help PF12 End ]

```

```
*** Field 39:
[-----]

*** Field 40:
[ ]

*** Field 41:
[ ]

*** Field 42:
[ ]

*** Field 43:
[ ]

*** Field 44:
[ ]

...*** Field 45:
[ ]

*** Field 46:
[ ]

*** Field 47:
[==>]

*** Field 48:
[ ]

*** Field 49:
[ ]
```

2.5 ECS Subroutines

The ECS defines a set of subroutines with two types of parameters: input and output. Programmers must declare variables of these same types in order to pass them to the subroutines.

In the following example, the program reads a group name contained in the third field of the 3270/5250 screen buffer, and branches off, depending on the contents of the output string FieldContents.

```
# This include file contains definitions of all ECS subroutines,
# constants, and a stimulus declaration.

include "eslmsrv.inc"

# Variable Declarations

# Integer variable FieldNumber will contain number of particular
# field, and string variable FieldContents will possess information
# contained in that field.

integer variable   FieldNumber
string variable   FieldContents
:
# Object Declarations
:
response to start
:
# We are interested in information contained in field #3. By
# passing this input to EcsReadField, we will get the contents of a
# field read into FieldContents variable, which we can manipulate
# after this call to DLL. In this example, we will invoke different
# programs, depending on the contents of field #3.

copy 3 to FieldNumber
call EcsReadField( FieldNumber, FieldContents )
switch FieldContents is
    case char "Development" is
        change to program "MainDv"
    case char "Support" is
        change to program "MainS"
    case line "Documentation" is
        change to program "MainDc"
```

```
case line "QA" is
    change to program "MainQA"
default is
    send "ERROR has occurred\n" to errorlog
end switch
```

2.5.1 ECS Subroutines and EASEL OS/2 EE Built-in Functions

All ECS subroutines, except the following, reset all EASEL OS/2 EE response inquiry built-in functions:

EcsClearWatch	EcsWatchForIIOff
EcsWatchForCursorAt	EcsWatchForStringAt
EcsWatchForCursorNotAt	EcsWatchForNotStringAt

For example:

response to Key_X

```
# This sequence will not make desired object
# green, because "object" was reset by
# EcsPressENTER subroutine.

call EcsPressENTER()
make object green
```

2.5.2 Summary of ECS Subroutines

The following table summarizes the subroutines provided in the ECS. Following the summary, each subroutine is described in detail. Each description gives the subroutine's declaration and input and output parameters, notes error conditions, gives sample code where applicable, and refers you to associated subroutines.

Subroutine	See Page	Usage
<i>Connecting and Disconnecting Sessions:</i>		
EcsConnect	2-34	Connects to a specified session.
EcsDisconnect	2-37	Disconnects from an active connected session.
<i>Examining the Buffer:</i>		
EcsReadField	2-40	Field-oriented. Copies the contents of the specified field into a string variable.

Subroutine	See Page	Usage
EcsGetFieldCount	2-43	Field-oriented. Copies the number of fields on the current screen into an integer variable.
EcsGetFieldLength	2-45	Field-oriented. Copies the length of a specified field into an integer variable.
EcsGetFieldType	2-47	Field-oriented. Copies the field type of a specified field into an integer variable.
EcsGetFieldAttribute	2-49	Field-oriented. Copies the state of the specified field attribute of the specified field into an integer variable.
EcsGetCursorPosition	2-51	Copies the row and column of the current cursor position into integer variables.
EcsGetFieldNumber	2-53	Field-oriented. Locates the 3270/5250 field number for a specified screen position.
EcsGetFieldPosition	2-55	Field-oriented. Copies the starting row and column position of a specified field into integer variables.
EcsReadLine	2-56	Line-oriented. Copies the contents of a line into a string variable.
EcsGetLastNonBlankLineNumber	2-58	Line-oriented. Copies the number of the last nonblank line at or above a specified starting line into an integer variable.
EcsReadScreen	2-60	Copies the contents of the 3270/5250 screen buffer into a textual region.
EcsReadColoredScreen	2-62	Copies the contents of the 3270/5250 screen buffer, with colors, into a colored textual region.

Subroutine	See Page	Usage
<i>Synchronization:</i>		
EcsClearWatch	2-67	Clears or erases a specified watch string.
EcsWatchForScreenSize	2-68	Watches for the number of rows and columns of the next screen, usually associated with changing 3270 Emulator screen size among different screens.
EcsWatchForCursorAt	2-71	Queues a command for watching the 3270/5250 screen until the cursor is located at a specified row and column position in the specified watch command.
EcsWatchForCursorNotAt	2-73	Queues a command for watching the 3270/5250 screen until the cursor is located anywhere except a specified row and column position in the specified watch command.
EcsWatchForIIOff	2-75	Queues a command for watching the Operator Information Area until the Input Inhibited indicator or X remains off for the period of time specified by input variable.
EcsWatchForStringAt	2-77	Queues a command for watching the 3270/5250 screen until a specified string appears at a specified row and column position in the specified watch command.
EcsWatchForStringNotAt	2-80	Queues a command for watching the 3270/5250 screen until any string other than a specified string appears at a specified row and column position in the specified watch command.

Subroutine	See Page	Usage
EcsWatch	2-82	Executes previously queued “watch” command(s) and activates a response you have defined to signal when a new screen is ready. Allows EASEL OS/2 EE to take other responses while “watch” is in effect. After this subroutine executes, you must use EcsConnect to reestablish a connection.
EcsWatchAndWait	2-85	Executes previously queued “watch” command(s) and waits until a new screen is ready before allowing EASEL OS/2 EE to take other responses.
<i>Communicating with the Host:</i>		
EcsSetAutoWait	2-87	Sets the user defined variable EcsAutoWaitFlag to control the AutoWait feature.
EcsPressKey	2-89	Emulates pressing a specified “special” key on a 3270/5250 terminal.
EcsPressENTER	2-92	Emulates pressing the ENTER key on a 3270/5250 terminal.
EcsTypeString	2-94	Simulates the typing of an ASCII string on a terminal keyboard.
EcsEnterString	2-96	Same as EcsTypeString followed by EcsPressENTER.
EcsWriteField	2-98	Simulates the typing of a string of data into a specified field.
<i>Controlling the Emulator Window:</i>		
EcsSetEmulatorWindow	2-100	Changes the state of the 3270 Emulator window.
EcsEmulate	2-102	Activates the 3270 Emulator window. Returns when you minimize the 3270 Emulator window.

Subroutine	See Page	Usage
EcsEmulateAndWatch	2-104	Similar to EcsEmulate. Returns when the 3270 Emulator displays a screen defined in a screen-ID file.
<i>Miscellaneous Subroutines:</i>		
EcsCheckIndicator	2-109	Inquires about the state of a particular indicator.
EcsGetSessionStatus	2-111	Copies the status of a 3270 session to which you are connected.
EcsGetSessionInfo	2-113	Copies the session type and screen dimensions of a specified session.
EcsGetError	2-115	Copies the error message associated with the specified error number into a string variable.
EcsGetCurrentSession	2-117	Provides user with facility to find current active session name.

2.5.3 Connecting and Disconnecting Sessions

2.5.3.1 EcsConnect

Description: EASEL OS/2 EE connects to the session specified by **SessionShortName** or any other user-defined string.

Input Variables: **SessionShortName** (string)

Output Variables: None

Declaration: subroutine EcsConnect(string:SessionShortName)
library "eslcmshr"

Remarks: This session becomes the active connected session. The previous active connected session, if any, remains connected, but it is no longer active. It is not an error to specify a session that is already connected; in fact, this is how you reactivate it.

SessionShortName must specify a 3270/5250 host session short name, configured under CM.

Example: The following example shows a use of the EcsConnect subroutine.

```
# This include file contains definitions for all ECS  
# subroutines and constants, as well as a stimulus  
# declaration.
```

```
include "eslcmshr.inc"
```

```
# Variable Definitions
```

```
:
```

```
# Object Definitions
```

```
:
```

```
response to start
```

```
# Initialize top, left, bottom, right margins of  
# a screen buffer, and pass name of a textual  
# region that will display contents of a screen  
# buffer to a variable.
```

```
copy 1 to Top
```

```
copy 1 to Left
```

```
copy 24 to Bottom
```

```
copy 80 to Right
```

```
copy Screen to TextRegion
```

```

# Initialize communications and connect to
# session F. If there are no errors, clear
# textual region Screen and read the
# contents of a screen buffer into it. If
# there is an error, exit EASEL OS/2 EE program.

copy "F" to SessionShortName
call EcsConnect( SessionShortName )
if ( errorlevel = 0 ) then
    clear Screen
    call EcsReadScreen(Top,Left,Bottom,Right,
        TextRegion)
else
    send "ERROR occurred... \n" to errorlog
    exit
end if
call EcsDisconnect()

# Initialize communications with new session B.
# If there are no errors, clear textual region
# Screen and read the contents of a screen
# buffer into it. If there is an error, exit
# EASEL OS/2 EE program. Session F is disconnected and
# inactive.

copy "B" to SessionShortName
call EcsConnect( SessionShortName )
clear Screen
if ( errorlevel = 0 ) then
    clear Screen
    call EcsReadScreen(Top,Left,Bottom,Right,
        TextRegion)
else
    send "ERROR occurred... \n" to errorlog
    exit
end if

# By specifying new SessionShortName "F", we are
# able to connect to another session. Session B
# is still connected but not active.

```

```

copy "F" to SessionShortName
call EcsConnect( SessionShortName )
clear Screen
if ( errorlevel = 0 ) then
    clear Screen
    call EcsReadScreen(Top,Left,Bottom,Right,
        TextRegion)
else
    send "ERROR occurred... \n" to errorlog
    exit
end if

```

<i>Errors:</i>	ECS_E_INUSE	Session is already in use.
	ECS_E_SESSNAME	Invalid session name.
	ECS_E_NOTINIT	Connection with a host is not established.

2.5.3.2 EcsDisconnect

Description: **EcsDisconnect** disconnects from the active connected session.

Input None

Variables:

Output None

Variables:

Declaration: subroutine EcsDisconnect()
 library "eslcmshr"

Remarks: To disconnect from a session that is connected but not active, you must first make it active by calling subroutine **EcsConnect**.

Example: The following example shows a use of the EcsDisconnect subroutine.

```
# This include file contains definitions for all ECS
# subroutines and constants, as well as a stimulus
# declaration.

include "eslcmshr.inc"

# Variable Definitions
:
# Object Definitions
:
response to start

# Initialize top, left, bottom, right margins of a
# screen buffer. Pass name of a textual region that
# will display contents of a screen buffer to a
# variable.

copy 1 to Top
copy 1 to Left
copy 24 to Bottom
copy 80 to Right
copy Screen to TextRegion

# Initialize communications and connect to session
# F. If there are no errors, clear textual region
# Screen and read the contents of a screen buffer
# into it. If there is an error, exit EASEL OS/2 EE
# program.
```

```

copy "F" to SessionShortName
call EcsConnect( SessionShortName )
if ( errorlevel = 0 ) then
    clear Screen
    call EcsReadScreen( Top,Left,Bottom,Right,
        TextRegion)
else
    send "ERROR occurred... \n" to errorlog
    exit
end if
call EcsDisconnect()
# Disconnects from current session F.
# Initialize communications with new session B.
# If there are no errors, clear textual region
# Screen and read the contents of a screen
# buffer into it. If there is an error, exit
# EASEL OS/2 EE program. Session F is disconnected and
# inactive.

copy "B" to SessionShortName
call EcsConnect( SessionShortName )
clear Screen
if ( errorlevel = 0 ) then
    clear Screen
    call EcsReadScreen( Top,Left,Bottom,Right,
        TextRegion)
else
    send "ERROR occurred... \n" to errorlog
    exit
end if

# By specifying new SessionShortName "F", we are
# able to connect to another session. Session B
# is still connected, but not active.

copy "F" to SessionShortName
call EcsConnect( SessionShortName )
clear Screen
if ( errorlevel = 0 ) then
    clear Screen
    call EcsReadScreen( Top,Left,Bottom,Right,
        TextRegion) else

```



```

end if
call EcsDisconnect()
# Disconnects from current session F.
# To disconnect from connected, but inactive,
# session, you must make it active by
# reconnecting with EcsConnect and then
# disconnect with EcsDisconnect subroutines.

copy "B" to SessionShortName
call EcsConnect( SessionShortName )
call EcsDisconnect()

# Disconnects from current session B.

```

```
exit
```

<i>Errors:</i>	ECS_E_NOTCONN	Not connected to an active session.
	ECS_E_NOTINIT	Connection with a host is not established.

Example: The following example shows a use of the EcsReadField subroutine.

```
# This include file contains definitions for all ECS
# subroutines and constants, as well as a stimulus
# declaration.

include "eslcmrv.inc"

# Variable Definitions
:
# Object Definitions
:
response to start
:
# To find out the contents of field #4, pass the
# argument to EcsReadField and make a call to DLL.
# The output will be contained in FieldContents
# and can be manipulated accordingly.

copy 4 to FieldNumber
call EcsReadField( FieldNumber,FieldContents )
add to Date_Window
    insert FieldContents
:
```

<i>Errors:</i>	ECS_E_FLDNUM	Invalid field number.
	ECS_E_NOFLDS	Screen contains no fields.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to active session.

2.5.4.2 EcsGetFieldCount

Description: **EcsGetFieldCount** copies the number of fields of current screen in the buffer into an integer variable FieldCount.

Input None

Variables:

Output **FieldCount** (integer)
Variables:

Declaration: subroutine EcsGetFieldCount(integer:FieldCount)
library "eslcsrv"

Example: The following example shows a use of the EcsGetFieldCount subroutine.

```
# This include file contains definitions for all ECS
# subroutines and constants, as well as a stimulus
# declaration.

include "eslcsrv.inc"

# Variable Definitions
:
# Object Definitions
:
response to start
:
# Will attain number of fields on current screen
# through EcsGetFieldCount call to DLL. The output
# will be contained in FieldCount variable.

call EcsGetFieldCount( FieldCount )
copy 18 to FieldNumber
copy ECS_FILLWITHNULLS to FillSwitch
copy "12345678901" to String
if ( FieldNumber > FieldCount ) then
    send " ERROR has occurred ...\n" to errorlog
    exit
else
    call EcsWriteField( FieldNumber,String,
        FillSwitch )
end if

:
```

<i>Errors:</i>	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to active session.
	ECS_E_NOFLDS	Screen contains no fields.

2.5.4.3 EcsGetFieldLength

Description: **EcsGetFieldLength** copies the length of a specified field **FieldNumber** into output variable **FieldLength**. It copies zero if there is an error.

Input Variables: **FieldNumber** (integer)

Output Variables: **FieldLength** (integer)

Declaration: subroutine EcsGetFieldLength(integer:FieldNumber,
integer:FieldLength)
library "eslcmrv"

Example: The following example shows a use of the **EcsGetFieldLength** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmrv.inc"

# Variable Declarations
:

# Object Declarations
:
response to start

# Initialize communications and connect to
# session F. Pass particular field number
# to a subroutine. Output is returned in
# FieldLength variable.

copy "F" to SessionShortName
call EcsConnect( SessionShortName )
copy 1 to FieldNumber
call EcsGetFieldLength(FieldNumber, FieldLength)
if ( FieldLength != 20 ) then
    send " ERROR has occurred...\n" to errorlog
else
    send " Field #1 : Length => passed...\n" to
        errorlog
end if
:
```

<i>Errors:</i>	ECS_E_FLDNUM	Invalid field number.
	ECS_E_NOFLDS	Screen contains no fields.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to active session.

2.5.4.4 EcsGetFieldType

Description: **EcsGetFieldType** copies an integer that specifies the field type into the variable **FieldType**.

Input Variables: **FieldNumber** (integer)

Output Variables: **FieldType** (integer)

Declaration: subroutine EcsGetFieldType(integer:FieldNumber,
integer:FieldType)
library "eslcsrv"

Remarks: It sets the above output integer variable, **FieldType**, to a value that denotes the type of data that may be contained in the field. (See 2.3.3, "Field-Oriented and Line-Oriented Screens" on page 2-16 for more information about field types.) Use the following constants to test the value of **FieldType**:

ECS_FT_ALPHANUMERIC *
ECS_FT_ALPHA_ONLY
ECS_FT_NUMERIC_SHIFT
ECS_FT_NUMERIC_ONLY *
ECS_FT_RESERVED
ECS_FT_DIGITS_ONLY
ECS_FT_MAG_STRIPE
ECS_FT_SIGNED_NUMERIC

* **Note:** These are the only valid field types for a 3270 session.

If there is an error, ECS will return an error event and **FieldType** will be set to -1.

Example: The following example shows a use of the **EcsGetFieldType** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"
# Variable Definitions
:
# Object Definitions
:
response to start
:
# To find TYPE of any field, pass the field
# number to input variable FieldNumber and
# make a call to DLL. Output will be returned
# in FieldType variable.

copy 32 to FieldNumber
call EcsGetFieldType( FieldNumber, FieldType )
if ( FieldType = ECS_FT_ALPHANUMERIC ) then
    send "Data type is ALPHANUMERIC\n" to errorlog
else
    send "ERROR has occurred...\n" to errorlog
end if
:
```

<i>Errors:</i>	ECS_E_FLDNUM	Invalid field number.
	ECS_E_NOFLDS	Screen contains no fields.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to active session.

2.5.4.5 EcsGetFieldAttribute

Description: **EcsGetFieldAttribute** copies the state of the specified field attribute of the specified field into an integer variable.

Input **FieldNumber** (integer)

Variables: **FieldAttribute** (integer)

Output **AttributeState** (integer)

Variables:

Declaration: subroutine EcsGetFieldAttribute(integer:FieldNumber,
 integer:FieldAttribut
 integer:AttributeStat

 library "eslcmshr"

Remarks: **EcsGetFieldAttribute** must be called separately for each field Attribute you are interested in. If you want all four parameters, you must call this subroutine four separate times.

Valid input for **FieldAttribute** includes:

ECS_PROTECTED
ECS_INTENSIFIED
ECS_NONDISPLAY
ECS_MODIFIED

Valid output for **AttributeState** includes:

ECS_NOT_PROTECTED
ECS_IS_PROTECTED
ECS_NOT_MODIFIED
ECS_IS_MODIFIED
ECS_NOT_NONDISPLAY
ECS_IS_NONDISPLAY
ECS_NOT_INTENSIFIED
ECS_IS_INTENSIFIED

Example: The following example shows a use of the **EcsGetFieldAttribute** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Definitions
:
# Object Definitions
:
response to start
:
# To find the ATTRIBUTE of any field, pass the
# field number to input variable FieldNumber,
# pass desired attribute to FieldAttribute,
# and make a call to DLL. Output will be
# returned in FieldType variable.

copy 32 to FieldNumber
copy ECS_PROTECTED to FieldAttribute
call EcsGetFieldAttribute( FieldNumber,
                          FieldAttribute,
                          AttributeState )
if ( AttributeState = ECS_IS_PROTECTED ) then
    send "Field 32 is protected\n" to errorlog
else
    send "ERROR:Field 32 is unprotected\n" to
        errorlog
end if
:
```

<i>Errors:</i>	ECS_E_FLDNUM	Invalid field number.
	ECS_E_NOFLDS	Screen contains no fields.
	ECS_E_FLDATTR	Invalid field attribute specified.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to active session.

2.5.4.6 EcsGetCursorPosition

Description: **EcsGetCursorPosition** returns current 3270/5250 screen cursor position row and column in output variables **Row** and **Column**.

Input None

Variables:

Output **Row** (integer)

Variables: **Column** (integer)

Declaration: subroutine EcsGetCursorPosition(integer:Row,
integer:Column)

library "eslcmshr"

Example: The following example shows a use of the **EcsGetCursorPosition** subroutine.

```
# This include file contains definitions for all ECS  
# subroutines and constants, as well as a stimulus  
# declaration.
```

```
include "eslcmshr.inc"
```

```
# Variable Definitions
```

```
:
```

```
# Object Definitions
```

```
:
```

```
response to start
```

```
:
```

```
# To find current cursor position, call  
# EcsGetCursorPosition. Output is retrieved  
# from Row and Column variables.
```

```
call EcsGetCursorPosition( Row, Column )
```

```
if (( Row != 6 ) and ( Column != 53 )) then  
    send "Returned values are incorrect\n" to  
    errorlog
```

```
else
```

```
    send "Returned values are correct\n" to  
    errorlog
```

```
end if
```

```
:
```

<i>Errors:</i>	ECS_E_NOTCONN	Not connected to active session.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_FLDNUM	Invalid field number.
	ECS_E_NOFLDS	Screen contains no fields.

See Also: **EcsGetFieldNumber**

2.5.4.7 EcsGetFieldNumber

Description: **EcsGetFieldNumber** finds the field number at the specified screen position. Screen positions start at (1,1) in the upper left corner of the screen.

Input **Row** (integer)
 Variables: **Column** (integer)

Output **FieldNumber** (integer)
 Variables:

Declaration: subroutine EcsGetFieldNumber(integer:Row,
 integer:Column,
 integer:FieldNumber)

 library "eslcmshr"

Remarks: If the given row (**Row**) and column (**Column**) are in the middle of a field, its field number is copied. If the given row and column are not inside a field, the number of the closest field farther down the screen is copied. If there is an error, **FieldNumber** contains zero.

Example: The following example shows a use of the **EcsGetFieldNumber** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmrv.inc"

# Variable Definitions
:
# Object Definitions
:
response to start
:
# To find the field number of any associated
# cursor position on a screen, use the
# EcsGetFieldNumber call to DLL. Pass particular
# screen coordinates to input variables Row and
# Column and make a call. Output will be
# returned in FieldNumber variable.

Copy 12 to Row
Copy 21 to Column
call EcsGetFieldNumber(Row, Column, FieldNumber)
if ( FieldNumber != 18 ) then
    send "Incorrect field: ERROR\n" to errorlog
else
    send "Correct field\n" to errorlog
end if
:
```

<i>Errors:</i>	ECS_E_NOFLDS	Screen contains no fields.
	ECS_E_ROWCOL	(Row, Column) is not on screen.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to active session.

See Also: **EcsGetCursorPosition**

2.5.4.8 EcsGetFieldPosition

Description: **EcsGetFieldPosition** copies the starting row **Row** and column **Column** position of a specified field **FieldNumber**. Position (1,1) is the upper left corner of the screen buffer.

Input Variables: **FieldNumber** (integer)

Output Variables: **Row** (integer)
Column (integer)

Declaration: subroutine EcsGetFieldPosition(integer:FieldNumber,
integer:Row,
integer:Column)

library "eslcsrv"

Errors: If there is an error, Row = 0, Column = 0.

Example: The following example shows a use of the **EcsGetFieldPosition** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.
```

```
include "eslcsrv.inc"
```

```
# Variable Definitions
```

```
:
```

```
# Object Definitions
```

```
:
```

```
response to start
```

```
:
```

```
copy 1 to FieldNumber
```

```
call EcsGetFieldPosition( FieldNumber, Row,  
Column )
```

```
if (( Row != 1 ) and ( Column != 2 )) then
```

```
send "Field #1 position -> FAIL\n" to  
errorlog
```

```
else
```

```
send "Field #1 position -> PASS\n" to  
errorlog
```

```
end if
```

```
:
```

<i>Errors:</i>	ECS_E_FLDNUM	Invalid field number.
	ECS_E_NOFLDS	Screen contains no fields.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to active session.

2.5.4.9 EcsReadLine

Description: **EcsReadLine** copies the contents of line number Row of the screen buffer into the string variable LineContents.

Input Variables: **Row** (integer)

Output Variables: **LineContents** (string)

Declaration: subroutine EcsReadLine(integer:Row,
string:LineContents)
library "eslcmshr"

Remarks: **EcsReadLine** is provided for use on screens that are line-oriented (unformatted). If it is applied to a screen that contains fields, the field information is ignored.

Reading the 25th Row of 5250 Sessions Under CM, the 5250 Work Station Feature supports a presentation space (screen buffer) size of 24 rows by 80 columns. In some instances, a 25th row is displayed. This occurs either when an error message from the host is displayed or when the operator presses the SysReq key.

You can read the contents of the 25th row by calling subroutine **EcsReadLine** after copying the value 25 to the input variable **Row**. If a 25th row is displayed currently, its contents are copied to the string variable LineContents. If a 25th row is not displayed, **EcsReadLine** returns with LineContents containing a null string ("").

EcsReadLine is the only ECS subroutine that allows you to specify row 25, and only for 5250 sessions.

For 3270 sessions, user may specify rows greater than 24 if and only if current screen size contains more rows. This may occur if 3270 Emulator is configured as Mod 3, Mod 4, or Mod 5.

Example: The following example shows a use of the **EcsReadLine** subroutine.

```
# This include file contains definitions for all  
# ECS subroutines and constants, as well as a  
# stimulus declaration.
```

```
include "eslcmshr.inc"
```

```
# Variable Declarations  
:
```

```

# Object Declarations
:
response to start
:
# Reads contents of line 3 into screen buffer.
# If error occurs, reads line 25 on 5250 screen
# and exits. The output is returned in
# LineContents variable.

copy 3 to Row
call EcsReadLine( Row, LineContents )
copy errorlevel to ErrorLevel
if ( ErrorLevel! = 0 ) then
    copy 25 to Row
    call EcsReadLine( Row, LineContents )
    send "5250 Error is " LineContents "\n" to
        errorlog
    exit
else
    switch LineContents is
        case char "Development" is
            change to program "MainDv"
        case char "Support" is
            change to program "MainS"
        case line "Documentation" is
            change to program "MainDc"
        case line "QA" is
            change to program "MainQA"
        default is
            send "ERROR has occurred\n" to errorlog
    end switch
end if
:

```

<i>Errors:</i>	ECS_E_LINENUM	Invalid line number.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to active session.

See Also: **EcsGetLastNonBlankLineNumber**

2.5.4.10 EcsGetLastNonBlankLineNumber

Description: **EcsGetLastNonBlankLineNumber** copies the number of the last nonblank line **LineNumber** on the screen at or above **StartLine**

Input Variables: **StartLine** (integer)

Output Variables: **LineNumber** (integer)

Declaration:

```
subroutine EcsGetLastNonBlankLineNumber
                                ( integer:StartLine,
                                integer:LineNumber )
library "eslcsrv"
```

Remarks: It copies the value zero if all lines at or above **StartLine** are blank or if there is an error. You must specify the initial value; the default is 0.

EcsGetLastNonBlankLineNumber is provided for use on screens that are line-oriented (unformatted), providing you with an easy way to locate the most recent prompt or message displayed by the application. If it is applied to a screen that contains fields, the field information is ignored.

Example: The following example shows a use of the **EcsGetLastNonBlankLineNumber** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcsrv.inc"

# Variable Declarations
:

# Object Declarations
:

# Subroutine Declarations

subroutine CheckOptions is
    # Interested in the contents of last line of
    # screen buffer that contains data. Pass value
    # to input variable and make a call to DLL.
    # Output is returned in LineNumber variable.
```

```

copy 24 to StartLine
call EcsGetLastNonBlankLineNumber( StartLine,
    LineNumber )
copy LineNumber to Row
call EcsReadLine( Row, LineContents )
switch LineContents is
    case char "Development" is
        change to program "MainDv"
    case char "Support" is
        change to program "MainS"
    case line "Documentation" is
        change to program "MainDc"
    case line "QA" is
        change to program "MainQA"
    default is
        send "ERROR has occurred\n" to errorlog
end switch
:
response to start
:
    call CheckOptions()
:

```

<i>Errors:</i>	ECS_E_LINENUM	Invalid line number.
	ECS_E_NOTCONN	Not connected to active session.
	ECS_E_NOTINIT	Connection with a host is not established.

See Also: **EcsReadLine**

2.5.4.11 EcsReadScreen

Description: **EcsReadScreen** inserts the contents of the 3270/5250 screen buffer into a textual region specified by variable **TextRegion**.

<i>Input</i>	Top	(integer)
<i>Variables:</i>	Bottom	(integer)
	Left	(integer)
	Right	(integer)
	TextRegion	(string)

Output Text from the screen buffer is inserted into the textual region **TextRegion**.

Definition: subroutine EcsReadScreen (integer:Top,
 integer:Left,
 integer:Bottom,
 integer:Right,
 string:TextRegion) is
 :
 :

Remarks: User must specify any rectangular area within the buffer by setting values for **Top**, **Bottom**, **Left**, and **Right**

You must clear the textual region before calling this subroutine, unless you want text inserted into the textual region at the current cursor position.

Example: The following example shows a use of the **EcsReadScreen** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.
```

```
include "eslcmshr.inc"
```

```
# Variable Declarations
```

```
:
```

```
# Object Declarations
```

```
:
```

```
enabled invisible white textual region Screen
  window size 80 columns 25 lines
  at position 0 95 in First
  black foreground
  title bar "EMULATOR SCREEN"
:
```

```

response to start
:
# Clear textual region Screen; initialize top,
# left, bottom, and right margins of screen
# buffer; and pass name of a region that will
# display screen buffer information to input
# variable TextRegion.

clear Screen
copy 3 to Top
copy 5 to Left
copy 20 to Bottom
copy 76 to Right
copy "Screen" to TextRegion

# If there are no errors, we will read contents
# of screen buffer into textual region Screen.
# Otherwise, will process the error.

if ( errorlevel = 0 ) then
    call EcsReadScreen( Top,Left,Bottom,Right,
        TextRegion )
else
    copy errorlevel to ErrorLevel
    clear Errors
    make Errors visible
    call EcsGetError( ErrorLevel,Message )
    add to Errors
        move to column 1 line 1
        insert "Error: " ErrorLevel
        move to column 1 line 3
        insert "ErrorText: " Message
    end if
:

```

<i>Errors:</i>	ECS_E_SCRNAREA	Invalid screen area limits.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to active session.

See Also: **EcsReadColoredScreen**

2.5.4.12 EcsReadColoredScreen

Description: **EcsReadColoredScreen** inserts the contents of the 3270/5250 screen buffer into a colored textual region specified by variable **ColoredTextRegion**.

<i>Input</i>	Top	(integer)
<i>Variables:</i>	Bottom	(integer)
	Left	(integer)
	Right	(integer)
	ColoredTextRegion	(string)

Output Colored text from the screen buffer is inserted into the colored textual region **ColoredTextRegion**.

Definition: subroutine EcsReadColoredScreen(integer:Top,
integer:Left,
integer:Bottom,
integer:Right,
string:ColoredTextRegion) is
:
:

Remarks: User must specify any rectangular area within the buffer by setting values for **Top**, **Bottom**, **Left**, and **Right**

You must clear the textual region before calling this subroutine, unless you want text inserted into the textual region at the current cursor position.

The colors that appear in the textual region are the same as the colors displayed by the 3270/5250 terminal emulation session, except in the following cases:

- Normal and intensified colors appear the same in the textual region. For example, the colors “cyan” and “light cyan” are distinguishable in the terminal emulator, but both appear as “cyan” in the colored textual region.
- The terminal emulator can simulate “underline” and “separator” field attributes by displaying an underline or some other symbol at character positions that actually contain nulls or blanks. The textual region never contains these simulated field attributes; it always contains the actual characters that reside in the screen buffer.

Example: The following example shows a use of the **EcsReadColoredScreen** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmrv.inc"

# Variable Declarations
:

# Object Declarations
:
enabled invisible colored textual region ColorScreen
  window size 80 columns 25 lines
  at position 0 95 in First
  black foreground
  title bar "COLOR EMULATOR SCREEN"
:
response to start
:
  # Clear colored textual region Screen;
  # initialize top, left, bottom, and right
  # margins of screen buffer; and pass name
  # of a region that will display screen buffer
  # information to input variable
  # ColoredTextRegion.

  clear ColorScreen
  copy 3 to Top
  copy 5 to Left
  copy 20 to Bottom
  copy 76 to Right
  copy "ColorScreen" to TextRegion

  # If there are no errors, we will read contents
  # of screen buffer into colored textual region
  # Screen, otherwise will process the error.

  if ( errorlevel = 0 ) then
    call EcsColoredReadScreen( Top,Left,Bottom,
      Right,ColoredTextRegion )
```



```

else
  copy errorlevel to ErrorLevel
  clear Errors
  make Errors visible
  call EcsGetError( ErrorLevel,Message )
  add to Errors
    move to column 1 line 1
    insert "Error: " ErrorLevel
    move to column 1 line 3
    insert "ErrorText: " Message
  end if
:

```

<i>Errors:</i>	ECS_E_SCRNAREA	Invalid screen area limits.
	ECS_E_NOTCONN	Not connected to active session.
	ECS_E_NOTINIT	Connection with a host is not established.

See Also: **EcsReadScreen**

2.5.5 Synchronizing 3270/5250 Support with the Host

Host transmissions to the screen buffer take time. If EASEL OS/2 EE examines the screen buffer too soon, it may try to perform subroutines based on an incomplete screen. The subroutines described in this section allow your program to synchronize with the host application so that the screen buffer is examined only when it contains a complete screen.

The ECS Watch Facility is the only group of subroutines available for host synchronization.

In order to use the watch facility, first define a sequence of one or more watch commands utilizing the following calls to DLL:

EcsClearWatch	EcsWatchForCursorAt
EcsWatchForScreenSize	EcsWatchForCursorNotAt
EcsWatchForStringAt	EcsWatchForIIOff
EcsWatchForStringNotAt	

These commands define a set of conditions that must all be satisfied, in sequence, before the transmission of a screen is considered to be complete. Then, invoke subroutine **EcsWatch** or **EcsWatchAndWait**, which tells 3270/5250 Support for EASEL OS/2 EE to start watching for those conditions to be met. For example, you can define a watch command sequence that expresses the following:

Watch the cursor until it is not located at position row 24, column 6.
Next, watch the Operator Information Area until the Input Inhibited indicator is off. Let me know when this is done.

(This is the example shown in the description of the **EcsWatchAndWait** subroutine.)

You can define watch commands to monitor any of the following:

- Change of screen size
- The contents of a string at a certain position within the screen buffer
- The cursor position
- The Input Inhibited indicator (II) in the Operator Information Area, or the "X" status.

You can tailor the watch command sequences to the way your host operating system and application update the terminal screens. It will be easier for you to define appropriate watch command sequences by first answering the following questions on how the screens are updated, and keeping in mind that **EcsWatchForIIOff** will be watching II or X indicators, depending upon which session type you are connected:

1. Does the screen size change when a new screen arrives?
2. When does the "II" or the "X" in the Operator Information Area appear and disappear? Does it flicker as characters are written to the screen?

Does it stay on continuously while waiting for the host to respond, or does it go on only when the host is transmitting?

3. Does the cursor position change from the previous screen? If so, when—after the “II” disappears, or just before?
4. Which characters will change from the previous screen? Which ones will stay the same? Will the screen be cleared before the next screen is displayed?

The answers to these questions will enable you to write EASEL OS/2 EE code that determines when the 3270/5250 screen buffer has been completely updated.

You define a watch command sequence by first calling the **EcsClearWatch** subroutine and then making one or more calls to the following subroutines:

EcsWatchForScreenSize	EcsWatchForIIOff
EcsWatchForCursorAt	EcsWatchForStringAt
EcsWatchForCursorNotAt	EcsWatchForStringNotAt

Each of these subroutines encodes the requested watch command by appending some characters to the global string variable **EcsWatchCommandString** or user defined watch string. These characters are interpreted by 3270/5250 Support for EASEL OS/2 EE at the time the watch commands are executed. (The contents of watch strings are managed by 3270/5250 Support for EASEL OS/2 EE, so it is not necessary to know how the commands are encoded.)

The subroutines **EcsWatch** and **EcsWatchAndWait** cause the current watch command sequence to be executed. Norm of these is called immediately after sending a keystroke that causes the host to transmit a new screen.

2.5.5.1 EcsClearWatch

Description: **EcsClearWatch** clears the specified watch string in preparation for defining a new sequence of watch commands to be executed by the **EcsWatch** or **EcsWatchAndWait** subroutines.

Input **WatchCommandString** (string)
Variables:

Output: None
Variables:

Definition: subroutine EcsClearWatch(string:WatchCommandString) is
 :
 :

Remarks: If a single behavior pattern defines the completion of screen transmission for all screens in the host application, then only a single **WatchCommandString** needs to be defined. Otherwise, it may be more convenient to define multiple **WatchCommandString** variables, one for each individual screen you expect to encounter. We suggest invoking **EcsClearWatch** for each defined **WatchCommandString**.

The global variable **EcsWatchCommandString** is internal to the ECS DLL and should be cleared and set when **EcsAutoWaitFlag** is set to **ECS_AUTOWAITON** because it is used by the AutoWait facility.

Example: See Examples for:

EcsWatchForScreenSize
EcsWatchForCursorAt
EcsWatchForCursorNotAt
EcsWatchForIIOff
EcsWatchForStringAt
EcsWatchForStringNotAt

2.5.5.2 EcsWatchForScreenSize

Description: **EcsWatchForScreenSize** watches for appearance of a new screen size specified by input variables **Rows** and **Columns**.

<i>Input Variables:</i>	Rows	(integer)
	Columns	(integer)
	WatchCommandString	(string)

Output Variables: None

Definition:

```
subroutine EcsWatchForScreenSize( integer:Rows,
                                integer:Columns,
                                string:WatchCommandString ) is
    :
```

Remarks: This subroutine is available only for 3270 Emulator sessions and should be used if user expects new screen to have different number of rows and columns.

Example: The following example shows a use of the **EcsWatchForScreenSize** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmrv.inc"

# Variable Declarations
:

# Object Declarations
:

# Action Declarations

action DefineLOGON is

    # This action will set up a watch string to look
    # for a new screen size of 27 rows and 132 columns,
    # cursor and string at specified positions.

    call EcsClearWatch( Watch1 )
    copy 27 to Row
    copy 132 to Column
```

```

call EcsWatchForScreenSize( Row, Column,
    Watch1 )
copy 39 to Row
copy 16 to Column
call EcsWatchForCursorAt( Row, Column, Watch1 )
copy 42 to Row
copy 2 to Column
copy "COMMAND" to String
call EcsWatchForStringAt( Row, Column, String,
    Watch1 )

```

action DefinePROFS is

```

# This action will set up a watch string to look
# for a new screen size of 24 rows and 80 columns,
# cursor and string at specified positions.

```

```

call EcsClearWatch( Watch1 )
copy 24 to Row
copy 80 to Column
call EcsWatchForScreenSize( Row, Column,
    Watch1 )
copy 42 to Row
copy 6 to Column
call EcsWatchForCursorAt( Row, Column, Watch1 )
copy 13 to Row
copy 8 to Column
copy "View main" to String
call EcsWatchForStringAt( Row, Column, String,
    Watch1 )

```

action Logout3270 is

```

# This action will log user out from PROFS and
# read LOGIN screen into text region.

```

```

action DefineLOGON
copy "logout" to String
call EcsEnterString( String )
wait 1
call EcsPressENTER()
copy 5000 to TimeLimit
call EcsWatchAndWait( TimeLimit, Watch1 )
if ( EcsWaitGaveUp ) then
    send "System ERROR, try again\n" to errorlog

```

```

else
    call EcsReadScreen( Top,Left,Bottom,Right,
                        TextRegion )
end if

response to start
# Initialize 3270 communications and connect to
# session A, which is MOD 5 ( 27 rows by 132
# columns ). Login into PROFS, changing screen
# size to 24 rows and 80 columns, and logout
# bringing back LOGIN screen defined by 27 rows
# and 132 columns.

copy "A" to SessionShortName
call EcsConnect( SessionShortName )
action DefinePROFS
copy 6 to FieldNumber
copy "vmuser78" to String
call EcsWriteField( FieldNumber, String,
                    FillSwitch )
copy "password" to String
copy 9 to FieldNumber
call EcsWriteField( FieldNumber, String,
                    FillSwitch )
call EcsPressENTER()
copy 15000 to TimeLimit
call EcsWatchAndWait( TimeLimit, Watch1 )
if ( EcsWaitGaveUp ) then
    send "System ERROR, try again\n" to errorlog
else
    call EcsReadScreen( Top,Left,Bottom,Right,
                        TextRegion )
end if
copy 5000 to TimeLimit
action Logout3270
call EcsDisconnect()
exit

```

<i>Errors:</i>	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_WATCH	Empty or invalid watch command string.

See Also: **EcsClearWatch**
 EcsWatch
 EcsWatchAndWait

2.5.5.3 EcsWatchForCursorAt

Description: **EcsWatchForCursorAt** watches the screen until the cursor is located at the position specified by **Row** and **Column**.

Input Variables: **Row** (integer)
Column (integer)

Output Variables: **WatchCommandString** (string)

Definition: subroutine EcsWatchForCursorAt(integer:Row,
integer:Column,
string:WatchCommandString) i
:

Remarks: **EcsWatchForCursorAt** does not get executed until a call is made to the **EcsWatch** or **EcsWatchAndWait** subroutine.

It is an error to have multiple WatchForCursorAt commands specified in the same WatchCommandString.

Example: The following example shows a use of the **EcsWatchForCursorAt** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmrv.inc"

# Variable Declarations

# Defines watch string variables Watch1 and Watch2.

string variable Watch1
                Watch2
:

# Object Declarations
:
```



```

response to start
:
# We will initialize watch string to watch for
# cursor at position (20,10) in screen buffer.
# First, initialize TimeLimit variable to select
# time you are willing to wait for a new screen.
# Then, clear watch string that you will be using
# with the EcsWatch or EcsWatchAndWait call with
# EcsClearWatch call to DLL. Initialize Row and
# Column (where you expect cursor to be on next
# screen) and set Watch1 string with
# EcsWatchForCursorAt call to DLL. When you call
# EcsWatchAndWait subroutine with TimeLimit and
# Watch1 input variables, you should wait for no
# more than 5 seconds. If cursor position is not
# (20,10) in screen buffer by that time,
# EcsWaitGaveUp variable will be set to "true".
# Otherwise, if new screen has arrived and cursor
# is in desired position within 5 seconds, we will
# read new screen information.

copy 5000 to TimeLimit
call EcsClearWatch( Watch1 )
copy 20 to Row
copy 10 to Column
call EcsWatchForCursorAt( Row, Column, Watch1 )
copy "logout" to String
call EcsEnterString( String )
call EcsWatchAndWait( TimeLimit, Watch1 )
if ( EcsWaitGaveUp ) then
    send " ERROR has occurred...\n" to errorlog
else
    call EcsReadScreen( Top,Left,Bottom,Right,
                        TextRegion )
    make Screen visible
end if
:

```

<i>Errors:</i>	ECS_E_WATCH	Empty or invalid watch command string.
	ECS_E_NOTINIT	Connection with a host is not established.

See Also: **EcsWatchForCursorNotAt**

2.5.5.4 EcsWatchForCursorNotAt

Description: **EcsWatchForCursorNotAt** watches the screen until the cursor is located at any position other than the one specified by **Row** and **Column**. In other words, it tests for the disappearance of the cursor from the specified position.

Input **Row** (integer)
Variables: **Column** (integer)

Output **WatchCommandString** (string)
Variables:

Definition: subroutine EcsWatchForCursorNotAt(integer:Row,
 integer:Column,
 string:WatchCommandString) is
 :
 :

Remarks: This command does not get executed until a call is made to the **EcsWatch** or **EcsWatchAndWait** subroutine.

Example: The following example shows a use of the **EcsWatchForCursorNotAt** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Declarations

# Defines watch string variables Watch1 and Watch2.

string variable Watch1
                  Watch2
:
:
# Object Declarations
:
:

response to start
:
:
# We will initialize watch string to watch for
# cursor to disappear from (20,10) position in
# screen buffer. First, initialize the TimeLimit
# variable to select time you are willing to
# wait for a new screen. Then, clear watch
# string that you will be using with the
```

```
# EcsWatch or EcsWatchAndWait call with
# EcsClearWatch call to DLL. Initialize Row and
# Column (where you expect cursor to disappear
# from on the screen) and set Watch1 string with
# EcsWatchForCursorNotAt call to DLL. When you
# call EcsWatchAndWait subroutine with TimeLimit
# and Watch1 input variables, you should wait
# for no more than 5 seconds. If cursor
# position is (20,10) in screen buffer by that
# time, EnsWaitGaveUp variable will be set to
# "true". Otherwise, if new screen has arrived
# and cursor is in new position within 5 seconds,
# we will read new screen information.
```

```
copy 5000 to TimeLimit
call EcsClearWatch( Watch1 )
copy 20 to Row
copy 10 to Column
call EcsWatchForCursorNotAt( Row, Column, Watch1 )
copy "logout" to String
call EcsEnterString( String )
call EcsWatchAndWait( TimeLimit, Watch1 )
if ( EcsWaitGaveUp ) then
    send " ERROR has occurred...\n" to errorlog
else
    call EcsReadScreen( Top,Left,Bottom,Right,
        TextRegion )
    make Screen visible
end if
:
```

<i>Errors:</i>	ECS_E_WATCH	Empty or invalid watch command string.
	ECS_E_NOTINIT	Connection with a host is not established.

See Also: **EcsWatchForCursorAt**

2.5.5.5 EcsWatchForIIOff

Description: **EcsWatchForIIOff** watches the Operator Information Area (OIA) until the Input Inhibited indicator ("II" for 5250, or "X" for 3270) is gone and remains gone for the period of time specified by **SettleTime**. (The Input Inhibited indicator is an "II" for 5250 sessions and "X" for 3270 sessions. They are equivalent and are hereafter referred to as simply the indicator.)

Input Variables: **SettleTime** (integer)

Output Variables: **WatchCommandString** (string)

Definition: subroutine EcsWatchForIIOff(integer:SettleTime,
string:WatchCommandString) is
:

Remarks: **SettleTime** is in units of .001 seconds (the value of 1000 is equivalent to 1 second). When the value of **SettleTime** is zero, **EcsWatchForIIOff** completes immediately once the indicator is off. Specifying a nonzero value for **SettleTime** is useful for host applications where the indicator flickers while the controller is writing data to the screen. Use of an appropriate value will ensure that the indicator is no longer flickering. If you do need to increase the value, it should be at least 200 (.20 seconds).

EcsWatchForIIOff does not get executed until a call is made to the **EcsWatch** or **EcsWatchAndWait** subroutine.

Example: The following example shows a use of the **EcsWatchForIIOff** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmrv.inc"

# Variable Declarations

# Defines watch string variables Watch1 and Watch2.
string variable Watch1
                Watch2

:
```

```

# Object Declarations
:
response to start
:
  # Initialize TimeLimit to 5 seconds and clear watch
  # string Watch2 with EcsClearWatch call to DLL.
  # Initialize SettleTime to 3 seconds and set watch
  # string Watch2 with EcsWatchForIIOff subroutine.
  # When EcsWatchAndWait is called, it will wait for
  # 3 seconds for II or X to be off in the status
  # line (in the period set by TimeLimit--5 seconds),
  # before setting the EcsWaitGaveUp variable. If II
  # or X were OFF for 3 seconds, EcsWaitGaveUp will
  # be set to "false", otherwise it will be "true".

copy 5000 to TimeLimit
call EcsClearWatch( Watch2 )
copy 3000 to SettleTime
call EcsWatchForIIOff( SettleTime, Watch2 )
copy "logout" to String
call EcsEnterString( String )
call EcsWatchAndWait( TimeLimit, Watch2 )
if ( EcsWaitGaveUp ) then
  send " ERROR has occurred...\n" to errorlog
else
  call EcsReadScreen( Top,Left,Bottom,Right,
                    TextRegion )
  make Screen visible
end if
:

```

<i>Errors:</i>	ECS_E_WATCH	Empty or invalid watch command string.
	ECS_E_NOTINIT	Connection with a host is not established.

2.5.5.6 EcsWatchForStringAt

Description: **EcsWatchForStringAt** watches the screen until the character string specified by **WatchString** appears at the location specified by **Row** and **Column**. In other words, **EcsWatchForStringAt** tests for the appearance of a specific character string at a specific location in the screen.

Input Variables:

Row	(integer)
Column	(integer)
WatchString	(string)

Output Variables:

WatchCommandString	(string)
---------------------------	----------

Definition:

```
subroutine EcsWatchForStringAt( integer:Row,
                               integer:Column,
                               string:WatchString,
                               string:WatchCommandString ) is
    :
```

Remarks: If **WatchString** contains more than one character, only the first character position is used to determine location of specified string in the screen buffer. The string should not contain any commas. **WatchString** can contain spaces between characters and can reach a maximum length of 24. If the length of **WatchString** is greater than 24 characters, the rest will be truncated. If you need to check for multiple strings/characters on the screen, you can set up multiple **EcsWatchForStringAt** definitions.

EcsWatchForStringAt does not get executed until a call is made to the **EcsWatch** or **EcsWatchAndWait** subroutine.

Example: The following example shows a use of the **EcsWatchForStringAt** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Declarations

# Defines watch string variables Watch1 and Watch2.

string variable Watch1
                Watch2
    :
```

```

# Object Declarations
:
response to start
:
# We will initialize Watch1 string to watch for
# string at position (20,10) in screen buffer.
# First, initialize TimeLimit variable to select
# time you are willing to wait for a new screen.
# Then, clear the watch string that you will be
# using with the EcsWatch or EcsWatchAndWait call
# with the EcsClearWatch call to DLL. Initialize
# Row and Column (where you expect string to be
# on next screen) and set Watch1 string with
# EcsWatchForStringAt call to DLL. Also,
# initialize String variable with expected
# character on the next screen. In this example,
# we will be watching for two characters to
# appear on next screen. When you call
# EcsWatchAndWait subroutine with TimeLimit and
# Watch1 input variables, you should wait for no
# more than 5 seconds. If characters don't appear
# in screen buffer at position (20,10) in that
# time, EcsWaitGaveUp variable will be set to
# "true". Otherwise, if new screen has arrived and
# characters are in desired position within 5
# seconds, we will read new screen information.

copy 5000 to TimeLimit
call EcsClearWatch( Watch1 )
copy 20 to Row
copy 10 to Column
copy "ABC Company" to String
call EcsWatchForStringAt( Row,Column,String,Watch1 )
copy 21 to Row
copy 40 to Column
copy "Budget for 1991" to String
copy "next" to String
call EcsEnterString( String )
call EcsWatchAndWait( TimeLimit, Watch1 )
if ( EcsWaitGaveUp ) then
    send " ERROR has occurred...\n" to errorlog
else
    call EcsReadScreen( Top,Left,Bottom,Right,
                        TextRegion )
    make Screen visible
end if
:

```

Errors: ECS_E_WATCH Empty or invalid watch command string.
 ECS_E_WATCHSTRING Empty WatchString was specified.
 ECS_E_NOTINIT Connection with a host is not established.

See Also: **EcsWatchForStringNotAt**

2.5.5.7 EcsWatchForStringNotAt

Description: **EcsWatchForStringNotAt** watches the screen until any character string other than the one specified by **WatchString** appears at the location specified by **Row** and **Column**. In other words, **EcsWatchForStringNotAt** tests for the disappearance of a specific character string at a specific location in the screen.

Input Variables:

WatchRow	(integer)
WatchCol	(integer)
WatchChar	(string)

Output Variables:

WatchCommandString	(string)
---------------------------	----------

Definition:

```
subroutine EcsWatchForStringNotAt( integer:Row,
                                   integer:Column,
                                   string:WatchString,
                                   string:WatchCommandString ) is
    :
```

Remarks: If **WatchString** contains more than one character, only the first character position is used to determine location of specified string in the screen buffer. The string should not contain any commas. **WatchString** may contain spaces between characters and can reach a maximum length of 24. If the length of **WatchString** is greater than 24 characters, the rest will be truncated. If you need to check for multiple strings/characters that are not on the screen, you can set up multiple **EcsWatchForStringNotAt** definitions.

EcsWatchForStringNotAt does not get executed until a call is made to the **EcsWatch** or **EcsWatchAndWait** subroutine.

Example: The following example shows a use of the **EcsWatchForStringNotAt** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmrv.inc"

# Variable Declarations
# Defines watch string variables Watch1 and Watch2.

string variable Watch1
                Watch2
    :
```

```

# Object Declarations
:

response to start
:
# We will initialize Watch1 string to watch for
# string to disappear from (20,10) position in
# screen buffer. Initialize the TimeLimit to
# select time you are willing to wait for a new
# screen. Clear watch string that you will be
# using with the EcsWatch or EcsWatchAndWait call
# with EcsClearWatch call to DLL. Initialize Row
# and Column (where you expect string to disappear
# from screen) and set Watch1 string with
# EcsWatchForStringNotAt call to DLL. Also,
# initialize String variable with character that
# will be gone on the next screen. In this
# example, we will be watching for two characters
# to disappear on next screen. When you call the
# EcsWatchAndWait subroutine with TimeLimit and
# Watch1 input variables, you should not wait for
# more than 5 seconds. If characters at the
# specified position in the screen buffer are
# still there by that time, EnsWaitGaveUp variable
# will be set to "true". If the new screen has
# arrived and the characters are gone within 5
# seconds, ECS will read new screen information.

copy 5000 to TimeLimit
call EcsClearWatch( Watch1 )
copy 20 to Row
copy 10 to Column
copy "ABC Company" to String
call EcsWatchForStringNotAt( Row,Column,String,
    Watch1 )
copy 21 to Row
copy 40 to Column
copy "Budget for 1991" to String
call EcsWatchForStringNotAt( Row,Column,String,
    Watch1 )

```

```

copy "next" to String
call EcsEnterString( String )
call EcsWatchAndWait( TimeLimit, Watch1 )
if ( EcsWaitGaveUp ) then
    send " ERROR has occurred...\n" to errorlog
else
    call EcsReadScreen( Top,Left,Bottom,Right,
                        TextRegion )
    make Screen visible
end if
:

```

<i>Errors:</i>	ECS_E_WATCH	Empty or invalid watch command string.
	ECS_E_WATCHSTRING	Empty WatchString was specified.
	ECS_E_NOTINIT	Connection with a host is not established.

See Also: **EcsWatchForStringAt**

2.5.5.8 EcsWatch

Description: **EcsWatch** executes the watch commands in **WatchCommandString** while allowing EASEL OS/2 EE to continue to respond to other stimuli that do not pertain to the screen buffer (such as object selections and keyboard input), as long as those responses are within a guarded or resumable begin/end block.

Input Variables: **WatchCommandString** (string)
TimeLimit (integer)

Output Variables: None

Declaration: subroutine EcsWatch(integer:TimeLimit,
string:WatchCommandString)
library "eslcmshr"

Remarks: The watch commands are executed one at a time, in the same order in which they were defined (via the **EcsWatchFor...** subroutines), until either there are no more watch commands left, or a period of time specified by **TimeLimit** has elapsed since **EcsWatch** was called. **TimeLimit** is in units of .001 seconds (the value 1000 is equivalent to 1 second).

To detect the completion of **EcsWatch**, you must set up the following **response** to **stimulus** statements within your main EASEL OS/2 EE application. Bear in mind that if the current state of execution of the EASEL OS/2 EE program is not within a guarded block, you might miss the response events. All three responses must be defined exactly as shown:

```
response to stimulus EaselComServer ECS_WATCHERROR
response to stimulus EaselComServer ECS_WATCHDONE
response to stimulus EaselComServer ECS_WATCHGAVEUP
```

ECS_WATCHDONE, **ECS_WATCHGAVEUP**, and **ECS_WATCHERROR** are status messages from DLL. Normally, **ECS_WATCHDONE** is received, unless **TimeLimit** elapses before all of the watch commands complete or an error has occurred.

We strongly recommend that you define the block as either guarded or resumable. If you want to restrict valid responses to be only those within the block, make the block guarded. If you want to have all other responses in outer blocks valid as well, make the block resumable. The begin/end block must have at least one **leave block** statement.

The **EcsWatch** subroutine returns immediately after being called, but the watch commands continue executing until one of the above status messages is received from the ECS. Calling any ECS subroutine while a watch is in progress interrupts and cancels the watch, and the status message **ECS_WATCHERROR** is sent to EASEL OS/2 EE.

In addition, each time **EcsWatch** is called, you must reconnect to the session with **EcsConnect** before you leave a block.

Example: The following example shows a use of the **EcsWatch** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmrv.inc"

# Variable Declarations

# Defines watch string variables Watch1 and Watch2.

string variable Watch1
                Watch2
:

# Object Declarations
:

response to start
:
# Clear string Watch2 and setup next watch string
# with EcsWatchForCursorAt. Initialize session and
# connect to the host. Initialize TimeLimit to 5
# seconds and send PF12 key to the host. Will call
# EcsWatch within guarded block, which defines
# available responses within that block. With each
# response to stimulus EaselComServer, you must
# first perform reconnect to the current session.

call EcsClearWatch( Watch2 )
copy 20 to Row
copy 16 to Column
call EcsWatchForCursorAt( Row,Column,Watch2 )
copy "A" to SessionShortName
call EcsConnect( SessionShortName )
copy 5000 to TimeLimit
copy ECS_PF12 to KeyCode
```

```

call EcsPressKey( KeyCode )
make Icons visible
call EcsWatch( TimeLimit,Watch2 )
begin guarded
  response to First
    make Icons yellow
  response to Icons
    make Icons red
    make object yellow
  response to stimulus EaselComServer
    ECS_WATCHGAVEUP
    call EcsConnect( SessionShortName )
    make Icons invisible
    send " ERROR has occurred... \n" to errorlog
    leave block
  response to stimulus EaselComServer
    ECS_WATCHERROR
    call EcsConnect( SessionShortName )
    make Icons invisible
    send " ERROR has occurred... \n" to errorlog
    leave block
  response to stimulus EaselComServer
    ECS_WATCHDONE
    call EcsConnect( SessionShortName )
    make Icons invisible
    call EcsReadScreen( Top,Left,Bottom,Right,
                        TextRegion )
:
end
:
```

<i>Errors:</i>	ECS_E_WATCH	Empty or invalid watch command string.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to an active session.
	ECS_E_MULTWATCH	Multiple watches are not currently supported.
	ECS_E_BADTIME	Illegal time limit specified. (0 < SettleTime < 1 hour)

See Also: **EcsWatchAndWait**
 EcsSetAutoWait

2.5.5.9 EcsWatchAndWait

Description: **EcsWatchAndWait** executes the watch commands in **WatchCommandString** in the same way as **EcsWatch**, except that it does not allow EASEL OS/2 EE to respond to other stimuli while **EcsWatchAndWait** is running.

Input Variables: **WatchCommandString** (string)
TimeLimit (integer)

Output Variables: **EcsWaitGaveUp** (boolean)

Definition: subroutine EcsWatchAndWait(integer:TimeLimit,
string:WatchCommandString) is
:

Remarks: The watch commands are executed one at a time, in the same order in which they were defined (via the **EcsWatchFor...** subroutines), until either there are no more watch commands left, or a period of time specified by **TimeLimit** has elapsed since **EcsWatchAndWait** was called.

The global boolean variable **EcsWaitGaveUp** will be set to "true" if the TimeLimit was exceeded before all watch conditions were satisfied, or if the watch sent back the **ECS_WATCHERROR** message. Otherwise, the watch conditions were all satisfied within the specified TimeLimit and **EcsWaitGaveUp** is set to "false."

Example: The following example shows a use of the **EcsWatchAndWait** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Declarations

# Defines watch string variables Watch1 and Watch2.

string variable Watch1
                Watch2
:

# Object Declarations
:
```

```

response to start
:
# Clear string Watch2 and set up next watch
# string with EcsWatchForCursorAt. Initialize
# session and connect to the host. Initialize
# TimeLimit to 5 seconds and send PF12 key to
# the host. Will call EcsWatchAndWait. If
# new screen doesn't arrive within 5 seconds,
# ECS will send error to errorlog.

call EcsClearWatch( Watch2 )
copy 20 to Row
copy 16 to Column
call EcsWatchForCursorAt( Row,Column,Watch2 )
copy "A" to SessionShortName
call EcsConnect( SessionShortName )
copy 5000 to TimeLimit
copy ECS_PF12 to KeyCode
call EcsPressKey( KeyCode )
call EcsWatchAndWait( TimeLimit,Watch2 )
if ( EcsWaitGaveUp ) then
    send " ERROR has occurred ...\n" to errorlog
else
    clear ColorScreen
    call EcsReadColoredScreen( Top,Left,Bottom,
                                Right,ColoredTextRegion )
end if
:

```

<i>Errors:</i>	ECS_E_WATCH	Empty or invalid watch command string.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to an active session.
	ECS_E_MULTWATCH	Multiple watches are not currently supported.
	ECS_E_BADTIME	Illegal time limit specified. (0 < SettleTime < 1 hour)

See Also: **EcsWatch**
 EcsSetAutoWait

2.5.6 Communicating With the Host

Host applications may require user responses. These responses can include keyed data, menu selections, and special function keys. 3270/5250 Support for EASEL OS/2 EE provides these capabilities through subroutines that simulate various keyboard functions and combinations of functions.

2.5.6.1 EcsSetAutoWait

Description: **EcsAutoWait** sets the global **EcsAutoWaitFlag** on or off, enabling or disabling an automatic call to **EcsWatchAndWait**, after data is transmitted to the host.

Input Variables: **AutoWaitFlag** (integer)

Output Variables: None

Definition: subroutine EcsSetAutoWait(integer:AutoWaitFlag) is
:

Remarks: When **EcsAutoWaitFlag** is **ECS_AUTOWAITON**, **EcsWatchAndWait** is called using the values specified in the global variables **EcsTimeLimit** and **EcsWatchCommandString**, both of which should be set beforehand. Use the following constants to test the value of **AutoWaitFlag**:

ECS_AUTOWAITOFF
ECS_AUTOWAITON

Default for **EcsWatchCommandString** is "", and execution of **EcsWatchAndWait** will fail unless **EcsWatchCommandString** is defined before calling **EcsWatchAndWait**. Unless you redefined the **EcsTimeLimit**, its default value of 3000 (3 sec) will be used.

Example: The following example shows a use of the **EcsSetAutoWait** subroutine.

```
# This include file contains definitions for all  
# ECS subroutines and constants, as well as a  
# stimulus declaration.
```

```
include "eslcmshr.inc"
```

```
# Variable Declarations
```

```
# Defines watch string variables Watch1 and Watch2.
```

```

string variable Watch1
                Watch2
:
# Object Declarations
:

# Action & Subroutine Declarations
:
action Logout3270 is

    # First, clear EcsWatchCommandString by calling
    # EcsClearWatch subroutine. Set AutoWaitFlag ON
    # and initialize EcsTimeLimit to 23 seconds.
    # Initialize EcsWatchCommandString for next watch
    # and send PF12 key to the host. EcsWatchAndWait
    # subroutine will be called automatically; check
    # boolean variable EcsWaitGaveUp if new screen has
    # arrived.

    call EcsClearWatch( EcsWatchCommandString )
    copy ECS_AUTOWAITON to AutoWaitFlag
    call EcsSetAutoWait( AutoWaitFlag )
    copy 23000 to EcsTimeLimit
    copy 5000 to SettleTime
    call EcsWatchForIIOff( SettleTime,
                          EcsWatchCommandString )
    copy ECS_PF12 to Keycode
    call EcsPressKey( Keycode )
    if ( EcsWaitGaveUp ) then
        send " ERROR has occurred... \n" to errorlog
    else
        call EcsReadColoredScreen( Top,Left,Bottom,
                                   Right ColoredTextRegion )
    end if

response to start
:
    action Logout3270
:

```

Errors:	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to an active session.
	ECS_E_SENDKEY	Failed to send keystroke(s).

See Also: **EcsWatch**
 EcsWatchAndWait

2.5.6.2 EcsPressKey

Description: **EcsPressKey** simulates the pressing of a nonprinting key that has special significance, such as TAB, ENTER, etc.

Input Variables: **KeyCode** (integer)

Output Variables: None

Definition: subroutine EcsPressKey(integer:KeyCode) is
 :

Remarks: Keys common to both 3270 and 5250:

ECS_ATTN	ECS_DOWN_ARROW
ECS_BACK_TAB	ECS_DUP
ECS_CLEAR	ECS_ENTER
ECS_DBL_LEFT_ARROW	ECS_ERASE_EOF
ECS_DBL_RIGHT_ARROW	ECS_ERASE_INPUT
ECS_DELETE	

Keys unique to 3270:

ECS_CURSR_SEL	ECS_PF3
ECS_DEV_CNCL	ECS_PF4
ECS_FIELD_MARK	ECS_PF5
ECS_IDENT	ECS_PF6
ECS_PA1	ECS_PF7
ECS_PA2	ECS_PF8
ECS_PA3	ECS_PF9
ECS_PF1	ECS_PF10
ECS_PF2	

Keys unique to 5250:

ECS_BACK_SPACE	ECS_CMD18
ECS_CMD	ECS_CMD19
ECS_CMD1	ECS_CMD20
ECS_CMD2	ECS_CMD21
ECS_CMD3	ECS_CMD22
ECS_CMD4	ECS_CMD23
ECS_CMD5	ECS_CMD24
ECS_CMD6	ECS_DBL_DOWN_ARROW
ECS_CMD7	ECS_DBL_UP_ARROW
ECS_CMD8	ECS_END
ECS_CMD9	ECS_FIELD_EXIT
ECS_CMD10	ECS_FIELD_MINUS
ECS_CMD11	ECS_FIELD_PLUS
ECS_CMD12	ECS_HELP
ECS_CMD13	ECS_HEX
ECS_CMD14	ECS_INSERT_TOGGLE
ECS_CMD15	ECS_RECORD_BACKSPACE
ECS_CMD16	ECS_ROLL_DOWN
ECS_CMD17	ECS_ROLL_UP

Copy one of these string constants to KeyCode.

If **EcsAutoWaitFlag** is set to **ECS_AUTOWAITON**, the subroutine calls **EcsWatchAndWait** before returning using the values specified in the global variables **EcsTimeLimit** and **EcsWatchCommandString** to provide automatic synchronization. (See 2.5.5.9, "EcsWatchAndWait.") The default value for **EcsAutoWaitFlag** is **ECS_AUTOWAITOFF**. (See 2.5.6.1, "EcsSetAutoWait")

Example: The following example shows a use of the **EcsPressKey** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Declarations

# Defines watch string variables Watch1 and Watch2.
string variable Watch1
                Watch2
:

# Object Declarations
:
```

```

response to start
:
# First, clear EcsWatchCommandString by calling
# EcsClearWatch subroutine. Set AutoWaitFlag ON
# and initialize EcsTimeLimit to 23 seconds.
# Initialize EcsWatchCommandString for next watch
# and send PF12 key to the host. EcsWatchAndWait
# subroutine will be called automatically; check
# boolean variable EcsWaitGaveUp if new screen
# has arrived.

call EcsClearWatch( EcsWatchCommandString )
copy ECS_AUTOWAITON to AutoWaitFlag
call EcsSetAutoWait( AutoWaitFlag )
copy 23000 to EcsTimeLimit
copy 5000 to SettleTime
call EcsWatchForIIOff( SettleTime,
                      EcsWatchCommandString )
copy ECS_PF12 to Keycode
call EcsPressKey( KeyCode )
if ( EcsWaitGaveUp ) then
    send " ERROR has occurred... \n" to errorlog
else
    call EcsReadColoredScreen( Top,Left,Bottom,
                              Right ColoredTextRegion )
end if
:

```

<i>Errors:</i>	ECS_E_BADKEYCODE	Key code is unrecognized.
	ECS_E_SENDKEY	Failed to send keystroke(s).
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to an active session.

See Also: **EcsPressENTER**
 EcsTypeString
 EcsEnterString
 EcsWaitForIIOff
 EcsSetAutoWait
 EcsWatchAndWait

2.5.6.3 EcsPressENTER

Description: **EcsPressENTER** simulates the pressing of the ENTER key on a 3270/5250 terminal.

Input Variables: None

Output Variables: None

Definition: subroutine EcsPressENTER() is
:

Remarks: This operation can also be performed by calling **EcsPressKey** with KeyCode set to **ECS_ENTER**. **EcsPressENTER** is provided for convenience, since the ENTER key is used frequently for many applications.

If **EcsAutoWaitFlag** is set to **ECS_AUTOWAITON**, the subroutine calls **EcsWatchAndWait** before returning. It uses the values specified in the global variables **EcsTimeLimit** and **EcsWatchCommandString**, which must be set before the call is made, to provide automatic synchronization. The default value for **EcsAutoWaitFlag** is **ECS_AUTOWAITOFF**. (See **EcsSetAutoWait** and **EcsWatchAndWait**.)

Example: The following example shows a use of the **EcsPressENTER** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Declarations

# Defines watch string variables Watch1 and Watch2.

string variable Watch1
                Watch2
:

# Object Declarations
:
response to start
:
```

```
# First, clear EcsWatchCommandString by calling
# EcsClearWatch subroutine. Set AutoWaitFlag ON
# and initialize EcsTimeLimit to 23 seconds.
# Initialize EcsWatchCommandString for next
# watch and send data + ENTER key to the host.
# EcsWatchAndWait will be called automatically;
# check boolean variable EcsWaitGaveUp if new
# screen has arrived.
```

```
call EcsClearWatch( EcsWatchCommandString )
copy ECS_AUTOWAITON to AutoWaitFlag
call EcsSetAutoWait( AutoWaitFlag )
copy 23000 to EcsTimeLimit
copy 5000 to SettleTime
call EcsWatchForIIOff( SettleTime,
                      EcsWatchCommandString )
copy "New DATA" to String
call EcsTypeString( String )
call EcsPressENTER()
if ( EcsWaitGaveUp ) then
    send " ERROR has occurred... \n" to errorlog
else
    call EcsReadColoredScreen( Top,Left,Bottom,
                              Right,ColoredTextRegion )
end if
:
```

<i>Errors:</i>	ECS_E_BADKEYCODE	Key code is unrecognized.
	ECS_E_SENDKEY	Failed to send keystroke(s).
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to an active session.

See Also: **EcsPressKey**
 EcsTypeString
 EcsEnterString
 EcsWaitForIIOff
 EcsSetAutoWait
 EcsWatchAndWait

2.5.6.4 EcsTypeString

Description: **EcsTypeString** simulates the typing of a string of data on a 3270/5250 terminal keyboard.

Input Variables: **String** (string)

Output Variables: None

Declaration: subroutine EcsTypeString(string:String)
library "eslcmshr"

Remarks: **String** can contain any printing ASCII characters, including spaces and tabs. (An ASCII tab character \t translates to pressing the Tab key on a 3270/5250 terminal.)

Characters in the string are entered into the screen buffer starting at the current cursor location. If the string length equals or exceeds the field size, the consequences are identical to what would happen if the user entered the string directly from a 3270/5250 terminal keyboard; in this case, it is dependent on the 3270/5250 application.

After entering information, you must call **EcsPressENTER** to send the information to the host.

Example: The following example shows a use of the **EcsTypeString** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Declarations

# Defines watch string variables Watch1 and Watch2.

string variable Watch1
                Watch2
:

# Object Declarations
:
```



```
:  
# First, clear Watch1 watch string by calling  
# EcsClearWatch subroutine. AutoWaitFlag is OFF  
# and initialize TimeLimit to 3 seconds.  
# Initialize Watch1 string with data for next  
  
# watch and send data + ENTER key to the host.  
# EcsWatchAndWait will be called; check boolean  
# variable EcsWaitGaveUp if new screen has  
# arrived.
```

```
call EcsClearWatch( Watch1 )  
copy 3000 to TimeLimit  
copy 5000 to SettleTime  
call EcsWatchForIIOff( SettleTime,Watch1 )  
copy "New DATA" to String  
call EcsTypeString( String )  
call EcsPressENTER()  
call EcsWatchAnWait( TimeLimit, Watch1 )  
if ( EcsWaitGaveUp ) then  
    send " ERROR has occurred... \n" to errorlog  
else  
    call EcsReadColoredScreen( Top,Left,Bottom,  
                                Right,ColoredTextRegion )  
end if
```

```
:
```

Errors: ECS_E_NOTINIT

Connection with a host is not established.

See Also: [EcsPressKey](#)
[EcsPressENTER](#)
[EcsEnterString](#)
[EcsSetAutoWait](#)
[EcsWatchAndWait](#)

2.5.6.5 EcsEnterString

Description: **EcsEnterString** is similar to **EcsTypeString** except that the additional keystroke ENTER is sent automatically. This is slightly faster than using **EcsTypeString** followed by **EcsPressENTER**.

Input Variables: String (string)

Output Variables: None

Definition: subroutine EcsEnterString(string:String) is
:

Remarks: If **EcsAutoWaitFlag** is set to **ECS_AUTOWAITON**, the subroutine calls **EcsWatchAndWait** before returning. It uses the values specified in the global variables **EcsTimeLimit** and **EcsWatchCommandString**, which must be set before the call is made, to provide automatic synchronization. The default value for **EcsAutoWaitFlag** is **ECS_AUTOWAITOFF**. (See **EcsSetAutoWait** and **EcsWatchAndWait**.)

Example: The following example shows a use of the **EcsEnterString** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmrv.inc"

# Variable Declarations

# Defines watch string variables Watch1 and Watch2.

string variable Watch1
                Watch2
:
# Object Declarations
:
```

response to start

```
:
# First, clear EcsWatchCommandString by calling
# EcsClearWatch subroutine. Set AutoWaitFlag ON
# and initialize EcsTimeLimit to 23 seconds.
# Initialize EcsWatchCommandString for next watch
# and send data + ENTER key to the host.
# EcsWatchAndWait will be called automatically;
# check boolean variable EcsWaitGaveUp if new
# screen has arrived.

call EcsClearWatch( EcsWatchCommandString )
copy ECS_AUTOWAITON to AutoWaitFlag
call EcsSetAutoWait( AutoWaitFlag )
copy 23000 to EcsTimeLimit
copy 5000 to SettleTime
call EcsWatchForIIOff( SettleTime,
                      EcsWatchCommandString )
copy "New DATA" to String
call EcsEnterString( String )
if ( EcsWaitGaveUp ) then
    send " ERROR has occurred... \n" to errorlog
else
    call EcsReadColoredScreen( Top,Left,Bottom,
                              Right ColoredTextRegion )
end if
:
```

Errors: ECS_E_NOTINIT

Connection with a host is not
established.

See Also: **EcsPressENTER**
EcsTypeString
EcsSetAutoWait
EcsWaitForIIOff
EcsWatchAndWait

2.5.6.6 EcsWriteField

Description: **EcsWriteField** copies the string contained in **String** directly into the field specified by the variable **FieldNumber**.

Input **FieldNumber** (integer)
Variables: **String** (string)
FillSwitch (integer)

Output None
Variables:

Declaration: subroutine EcsWriteField(integer:FieldNumber,
string:String,
integer:FillSwitch)

library "eslcmshr"

Remarks: **FillSwitch** is a user-defined variable with one of the following values:

ECS_NOFILL
ECS_FILLWITHBLANKS
ECS_FILLWITHNULLS

ECS_NOFILL overwrites existing data with new data, and leaves the original field characters that were not overwritten by the input string untouched.

ECS_FILLWITHBLANKS and **ECS_FILLWITHNULLS** pad data with blanks or nulls to the end of a field.

(**ECS_FILLWITHNULLS** is not available for 5250 sessions.)

If **String** is longer than the field, trailing characters of **String** will be truncated.

Example: The following example shows a use of the **EcsWriteField** subroutine.

```
# This include file contains definitions for all  
# ECS subroutines and constants, as well as a  
# stimulus declaration.
```

```
include "eslcmshr.inc"
```

```
# Variable Declarations  
:
```

```
# Object Declarations  
:
```

response to start

```
:
# Specify particular field number, initialize
# desired values for FillSwitch and String, and
# write String into field #6. Now field #6
# contains string "12345678901234". By changing
# String value to "ABCD" and not changing
# FillSwitch, ECS will overwrite existing text
# of field #6. The resulting string will be
# "ABCD5678901234", which can be verified by
# reading the contents of a field or screen.

copy 6 to FieldNumber
copy ECS_NOFILL to FillSwitch
copy "12345678901234" to String
copy "ABCD" to String
call EcsWriteField( FieldNumber,String,
    FillSwitch )
if ( errorlevel = 0 ) then
    call EcsReadColoredScreen( Top,Left,Bottom,
                                Right,ColoredTextRegion )
else
    action KnownError
end if
:
```

<i>Errors:</i>	ECS_E_FLDNUM	Invalid field number.
	ECS_E_NOFLDS	Screen contains no fields or has not been scanned.
	ECS_E_PROTFLD	A protected field was specified.
	ECS_E_FILLSWITCH	An incorrect value for FillSwitch was specified.
	ECS_E_NOTINIT	Connection with a host is not established.

2.5.7 Controlling the Emulator Window

2.5.7.1 EcsSetEmulatorWindow

Description: **EcsSetEmulatorWindow** specifically controls the appearance and characteristics of a 3270 Terminal Emulator window through available parameters specified by the user-defined variable **Command**.

Input Variables: **Command** (integer)

Output Variables: None

Definition: subroutine EcsSetEmulatorWindow(integer:Command) is
:

Remarks: The characteristics of **EcsSetEmulatorWindow** are cumulative, meaning that you can pass more than one characteristic through the **Command** variable. See Example.

EcsSetEmulatorWindow status flags include:

ECS_EW_MAKE_VISIBLE
ECS_EW_MAKE_INVISIBLE
ECS_EW_ACTIVATE
ECS_EW_DEACTIVATE
ECS_EW_MINIMIZE
ECS_EW_MAXIMIZE
ECS_EW_RESTORE

The following combinations are illegal:

visible/invisible
activate/deactivate
minimize/maximize
restore/minimize
restore/maximize

Example: The following example shows a use of the **EcsSetEmulatorWindow** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Declarations
:
# Object Declarations
:

response to start
  copy "A" to SessionShortName
  call EcsConnect( SessionShortName )
  :
  # This will make 3270 Emulator Window invisible.

  copy ECS_EW_MAKE_INVISIBLE to Command
  call EcsSetEmulatorWindow( Command )
  :
  # This will make 3270 Emulator Window visible,
  # active, and OS/2 window ( not maximized ).

  copy ( ECS_EW_MAKE_VISIBLE + ECS_EW_RESTORE +
         ECS_EW_ACTIVATE ) to Command
  call EcsSetEmulatorWindow( Command )
  :
```

<i>Errors:</i>	ECS_E_NOT3270SESSION	Not connected to a 3270 session.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_BADCOMMAND	Bad emulate command.
	ECS_E_BADCOMBO	Bad emulate command combination.

2.5.7.2 EcsEmulate subroutine

Description: **EcsEmulate** causes the Emulator to take control of workstation, enabling you to "hot key" to the 3270 Emulator without having to enter any commands.

Input None
Variables:

Output None
Variables:

Definition: subroutine EcsEmulate() is
 :

Remarks: To exit the 3270 Emulator, you must first minimize it. This feature is available only for 3270 OS/2 emulation.

Example: The following example shows a use of the **EcsEmulate** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Declarations
:

# Object Declarations
:

response to start
  copy "A" to SessionShortName
  call EcsConnect( SessionShortName )

  # Will give control to 3270 Emulator. Control will
  # return to EASEL OS/2 EE application when 3270 Emulat
  # is minimized.

  call EcsEmulate()

  # This command will not be executed until 3270
  # Emulator is minimized.

  call EcsReadScreen( Top,Left,Bottom,Right,
                      TextRegion )

  :
```


<i>Errors:</i>	ECS_E_NOT3270SESSION	Not connected to a 3270 session.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to an active session.

See Also: **EmulateAndWatch**

2.5.7.3 EcsEmulateAndWatch

Description: **EcsEmulateAndWatch** causes the 3270 Emulator to relinquish control to an application, if it knows about a particular screen it encounters. The screen is defined by **ScreenIDFile**.

Input Variables: **ScreenIDFile** (string)

Output Variables: **ScreenName** (string)

Definition: subroutine EcsEmulateAndWatch (string:ScreenIDFile,
string:ScreenName) is
:

Remarks: **EcsEmulateAndWatch** is similar to action **EcsEmulate** in that it places the user in the windowed emulator by making it visible, active, and maximized. The difference is that **EcsEmulateAndWatch** allows you to specify one or more screens which, when encountered by the emulator, automatically cause the action to return.

EcsEmulateAndWatch allows you to effectively combine an EASEL OS/2 EE front-end with terminal emulation and switch between these. Using it, you can implement an EASEL OS/2 EE interface to a host application in an incremental fashion; those screens for which you have not yet written a graphical interface can be accessed by the user via the emulator. Unlike action **EcsEmulate**, **EcsEmulateAndWatch** lets you restrict the user to accessing only a particular set of screens while in the emulator. As soon as the emulator encounters a screen which your program knows about, **EcsEmulateAndWatch** returns and EASEL OS/2 EE regains control of the host application.

You define which screens cause EASEL OS/2 EE to regain control by creating a screen ID file. Copy the name of this file to a string variable (ScreenIDFile) before calling **EcsEmulateAndWatch**. A screen ID file must contain one or more sets of lines in the following format:

```
screen SCREEN_NAME
  ROW_NUM1    COL_NUM1    MATCH_STRING1
  ROW_NUM2    COL_NUM2    MATCH_STRING2
  ...         ...         ...
```

Each set of lines defines a screen that causes EASEL OS/2 EE to regain control from the emulator. SCREEN_NAME is any name or word you want to use to identify a screen. When EcsEmulateAndWatch returns, this name appears in the output variable ScreenName. When more than one screen is specified in the screen ID file, ScreenName allows you to easily determine which of the screens has caused EcsEmulateAndWatch to return and is, therefore, currently being displayed by the host application. If this action returns because the user has minimized the emulator window, ScreenName contains an empty string.

Each line beginning with **screen** must be followed by one or more lines containing a match string. These lines specify how the screen must appear. The screen is not recognized unless the string MATCH_STRING appears in the screen buffer, exactly as specified, starting at the location specified by ROW_NUM and COL_NUM. If multiple match-string lines are listed below a **screen** line, all of them must be satisfied in order for that screen to be recognized. MATCH_STRING must be delimited by double-quotation marks (as are EASEL OS/2 EE string literals). A double-quotation mark can appear inside MATCH_STRING if it is preceded by the EASEL OS/2 EE escape character (\). The MATCH_STRING should not contain any null characters. The ROW_NUM, COL_NUM, and MATCH_STRING must all be specified and listed on one line.

Upon returning, the emulator window is returned to the appearance it held previous to calling this action.

EcsEmulateAndWatch does not affect the appearance of windows belonging to the EASEL OS/2 EE program; we recommend that the primary window of the program be made invisible before calling **EcsEmulateAndWatch** and be made visible once it returns (see example).

Example: The following example shows a use of the **EcsEmulateAndWatch** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmrv.inc"

# Variable Declarations
:

# Object Declarations
:

# Action & Subroutine Declarations

action Logon is
  copy "login" to String
  call EcsEnterString( String )
  copy 15000 to TimeLimit
  call EcsWatchAndWait( TimeLimit, Watch1 )
  if ( EcsWaitGaveUp ) then
    send "ERROR: will try again \n" to errorlog
    call EcsDisconnect()
    exit
  else

    # Control will be gained by EASEL OS/2 EE or 3270
    # Emulator based on arrival of a new screen
    # into screen buffer and the screen
    # information contained in ScreenIDFile.
    # ECS expects screen buffer to contain
    # PROFS main screen. If it doesn't, user
    # defined error condition will occur.
    # Otherwise, appropriate action will be
    # called.

    call EcsEmulateAndWatch( ScreenIDFile,
                             ScreenName )
    if (ScreenName != "First1") then
      send "Error - not at Profs\n" to errorlog
      call EcsDisconnect()
      exit
    else
      action ScreenName
    end if
  end if
  :
```

```

response to start
:
# Pass the name of a file containing screen
# information to a variable ScreenIDFile. By
# calling EcsEmulateAndWatch, ECS will decide
# which application will have control, EASEL OS/2 EE
# vs. 3270 Emulator, based on the screen
# information contained in ScreenIDFile.

copy "screens.dat" to ScreenIDFile
call EcsEmulateAndWatch( ScreenIDFile,
                        ScreenName)

# This call will recognize Login Screen of 3270
# Emulator, and action Logon will be called
# based on information in ScreenIDFile listed
# below.

if (ScreenName = "") then
    send "Error - not at Login Screen\n" to
        errorlog
    call EcsDisconnect()
    exit
else
    action ScreenName
end if
:
# This is a screen definition file.

# NOTE: ALL COMMENTS MUST BE EXCLUDED FROM ACTUAL FILE.

screen Logon                # Defined LOGIN screen.
1 2          "VIRTUAL"
18 2         "Fill"
24 61        "RUNNING"

screen First1               # Defined PROFS screen.
2 3          "Press one"
5 8          "Find"
7 3          "PF5"
23 1         "===>"

```

<i>Errors:</i>	ECS_E_SCRIDFILE	Screen ID file is missing.
	ECS_E_BADSCRID	Bad screen ID file.
	ECS_E_NOT3270SESSION	Not connected to a 3270 session.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to an active session.

See Also: **EcsEmulate**

2.5.8 Miscellaneous Subroutines

2.5.8.1 EcsCheckIndicator

Description: **EcsCheckIndicator** inquires about the state of a particular indicator.

Input Variables: **Indicator** (integer)

Output Variables: **IndicatorState** (integer)

Declaration: subroutine EcsCheckIndicator(integer:Indicator,
integer:IndicatorState)
library "eslcmshr"

Remarks: Possible values for indicators are as follows:

5250 Session

ECS_I_SA
ECS_I_II
ECS_I_MW

3270 Session

ECS_I_X

The state is returned in the variable **IndicatorState**. The returned value is an integer, depending on the input parameter. The following constants define the state of the indicators:

ECS_UNKNOWN	ECS_XUNAUTH
ECS_XREADY	ECS_XGOAWAY
ECS_XCLOCK	ECS_XMORE
ECS_XSYSTEM	ECS_XNUM
ECS_XMECHCK	ECS_XWHTNUM
ECS_XCOMMCK	ECS_XBADCRD
ECS_XPROGCK	ECS_XNOSYM
ECS_XWHAT	ECS_SAOFF
ECS_XNOFUN	ECS_SAON
ECS_XBADFUN	ECS_IIOFF
ECS_XKEY	ECS_IION
ECS_XNOPRT	ECS_MWOFF
ECS_XPRTBSY	ECS_MWON
ECS_XPRTVRB	

Example: The following example shows a use of the **EcsCheckIndicator** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Declarations
:

# Object Declarations
:

response to start

# Passing indicator of interest to variable
# Indicator and invoking ECS subroutine. Output
# value will be returned in IndicatorState
# variable; actions may be taken on this
# returned value.
copy ECS_I_SA to Indicator
call EcsCheckIndicator( Indicator,
                        IndicatorState )
if ( IndicatorState = ECS_SAON ) then
    send "Connection with the host established:
        session " SessionShortName "\n"
        to errorlog
else
    send " System is not available \n" to errorlog
    exit
end if
:
```

<i>Errors:</i>	ECS_E_BADINDICATOR	Indicator is unavailable for current session type.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to an active session.

2.5.8.2 EcsGetSessionStatus

Description: **EcsGetSessionStatus** returns a status of the current active session.

Input None

Variables:

Output **SessionStatusIndicator** (integer)

Variables:

Declaration: subroutine EcsGetSessionStatus
 (integer:SessionStatusIndicator)
library "eslcmshr"

Remarks: This call is valid for 3270 OS/2 emulator sessions, and will produce an error event if used for 5250 session.

Valid **SessionStatusIndicator** output is defined as follows:

ECS_SS_DISCONNECTED
ECS_SS_UNOWNED
ECS_SS_CONTROL
ECS_SS_APPLICATION

Example: The following example shows a use of the **EcsGetSessionStatus** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Declarations
:

# Object Declarations
:

response to start
:
# Will find out session status by making ECS call.
call EcsGetSessionStatus( SessionStatus )
if ( SessionStatus = ECS_SS_APPLICATION ) then
    send " SessionStatus: Owned by
        application -> \n" to errorlog
else
    send " GetSessionStatus ERROR ==>>> "
        SessionStatus "\n" to errorlog
end if
:
```

<i>Errors:</i>	ECS_E_NOSTATUS	Cannot determine status.
	ECS_E_NOT3270SESSION	Not connected to a 3270 session.
	ECS_E_NOTINIT	Connection with a host is not established.
	ECS_E_NOTCONN	Not connected to an active session.

2.5.8.3 EcsGetSessionInfo

Description: Subroutine **EcsGetSessionInfo** copies information about the session specified by **SessionShortName**.

Input Variables: **SessionShortName** (string)

Output Variables: **Type** (integer)
Rows (integer)
Columns (integer)

Declaration:

```
subroutine EcsGetSessionInfo
( string:SessionShortName,
  integer:Type,
  integer:Rows,
  integer:Columns )
library "eslcmshr"
```

Remarks: **SessionShortName** can contain the short session name of any started session, whether or not it is currently connected. When **EcsGetSessionInfo** returns, **Type** contains a value that represents the session type (see table below) and **Rows** and **Columns** contain the dimensions of the session's screen buffer. These values are usually 24 and 80, respectively, for ECS host sessions. They can change, however, for different 3270 modes.

Note that the value returned in **Rows** does not include the Operator Information Area. Also, the values returned in **Rows** and **Columns** are not meaningful for printer sessions.

The following integer constants must be used when testing the value of **Type**:

Integer Constants	Meaning
ECS_ST_5250_HOST	5250 host terminal session
ECS_ST_5250_PRINTER	5250 printer session
ECS_ST_3270_HOST	3270 host terminal session
ECS_ST_3270_PRINTER	3270 printer session
ECS_ST_PC	Session is local to the Programmable Workstation
ECS_ST_UNKNOWN	Session type cannot be determined

The following values are returned if an error (**ECS_E_SESSNAME**) occurs:

Variable	Value
Type	ECS_ST_UNKNOWN
Rows	0
Columns	0

Example: The following example shows a use of the **EcsGetSessionInfo** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcmshr.inc"

# Variable Declarations
:

# Object Declarations
:

response to start
:
# Can be checked in conjunction with ECS
# subroutine EcsWatchForScreenSize to make
# sure the MOD has changed for a new screen.
call EcsGetSessionInfo( SessionShortName,
                        Type,Rows,Columns )
if ( Type = ECS_ST_3270_HOST ) then
    send " Type 3270 HOST\n " to errorlog
else
    send "ERROR =====> NOT A 3270 HOST\n" to
    errorlog
end if
if (( Rows = 24 ) and ( Columns = 80 )) then
    send " Correct # of Rows/Columns\n" to
    errorlog
else
    send "ERROR =====> WRONG # of ROWS/COLUMNS\n"
    to errorlog
end if
:
```

Errors:	ECS_E_SESSNAME	Invalid session name.
	ECS_E_NOTINIT	Connection with a host is not established.

2.5.8.4 EcsGetError

Description: **EcsGetError** copies the error message associated with the specified error number into a string variable.

Input Variables: **ErrorLevel** (integer)

Output Variables: **ErrorMessage** (string)

Declaration: subroutine EcsGetError(integer:ErrorLevel,
string:Message)
library "eslcsrv"

Remarks: Ideally, **EcsGetError** should be checked after each call to the DLL.

Example: The following example shows a use of the **EcsGetError** subroutine.

```
# This include file contains definitions for all
# ECS subroutines and constants, as well as a
# stimulus declaration.

include "eslcsrv.inc"

# Variable Declarations
:

# Object Declarations
:

# Action & Subroutine Declarations

action KnownError is
    copy errorlevel to ErrorLevel
    if ( ErrorLevel != 0 ) then
        clear Errors
        make Errors visible
        call EcsGetError( ErrorLevel,Message )
        add to Errors
            move to column 1 line 1
            insert "Error: " ErrorLevel
            move to column 1 line 3
            insert "ErrorText: " Message
        wait 2
        make Errors invisible
        send " Error: " ErrorLevel "\n" to errorlog
    end if
```

response to start

:

Generally, this action should be called after
each call to DLL. It copies the current
errorlevel value to variable ErrorLevel and
passes it to DLL. It returns error text in
output variable Message, which is then
displayed with error level in the textual
region Errors.

action KnownError

:

Errors: ECS_E_NOTINIT

Connection with a host is not
established.

2.5.8.5 EcsGetCurrentSession

Description: Provides user with facility to poll current active session for its configured name.

Input Variables: None

Output Variables: **SessionShortName**

Declaration:

```
subroutine EcsGetCurrentSession
                                ( string:SessionShortName )
library "eslcsrv"
```

Remarks: This call is valid for both 3270 and 5250 sessions.

Example: The following example shows a use of the **EcsGetCurrentSession** subroutine.

```
# This file contains definitions for all ECS
# subroutines and constants, as well as a
# stimulus declaration.

include "eslcsrv.inc"

# Variable Declarations
:

# Object Declarations
:

response to start

# Initialize 3270 communications and connect to
# session A
copy "A" to SessionShortName
call EcsConnect( SessionShortName )
copy "" to SessionShortName
call EcsGetCurrentSession( SessionShortName )
if ( SessionShortName = "A" ) then
    send "Connected to Session A\n " to errorlog
else
    send "Connected to wrong session:
        " SessionShortName "\n" to
        errorlog
    exit
end if
:
```

Errors: , ECS_E_NOTINIT

Connection with a host is not established.

2.6 Error Handling

2.6.1 Error Status Retrieval

If the ECS encounters an error, processing is halted and the error level is set to the number of the error encountered. The value can be retrieved through the built-in function **errorlevel**. When no errors are detected, **errorlevel** contains the value zero. **EcsGetError** is a subroutine that sets a string variable to the error message associated with the error level passed to it.

```

:
call EcsPressENTER()
if ( errorlevel != 0 ) then
    copy errorlevel to ErrorLevel
    call EcsGetError( ErrorLevel, Message )
    send "Error is: " ErrorLevel "\n" to errorlog
    send "ErrorText is: " Message "\n" to errorlog
else
    call EcsReadScreen( Top,Left,Bottom,Right,
                        TextRegion)
    if ( errorlevel != 0 ) then
        copy errorlevel to ErrorLevel
        call EcsGetError( ErrorLevel, Message )
        send "Error is: " ErrorLevel "\n" to errorlog
        send "ErrorText is: " Message "\n" to errorlog
    end if
end if
:
```

In the previous example **ErrorLevel** contained the error code number that denotes the particular error that occurred, and **Message** contained the current error message.

If there are no errors present, **errorlevel** will contain the value 0, which equals **ECS_E_ERRORFREE**, with associated text "No errors detected." Each subroutine described in this publication lists the error conditions that can result when the subroutine is called, and shows the error code and associated error message text. The error code value is always represented by an integer constant whose names begin with **ECS_E_**. (See 2.7.2, "Reserved Constants" on page 2-130.) Always use these constants when testing the value of **errorlevel**. In general, you don't need to include statements in your program that check for all possible errors after every subroutine call. Many of the error conditions are caused by coding mistakes, as in the following example, and should never occur in a fully debugged program. Suppose the following EASEL OS/2 EE statements are executed:

```
# These statements will generate ECS error
# condition, because EcsReadLine cannot accept
# negative value for variable Row.
```

```
copy -3 to Row
call EcsReadLine( Row, LineContents )
```

After **EcsReadLine** returns, the errorlevel will contain **ECS_E_LINENUM** and associated error message “Invalid line number” can be retrieved with a **EcsGetError** call to DLL as described earlier.

2.6.2 Error Messages (Cross Reference)

2.6.2.1 No Errors Detected

Message: **No errors detected**
Constant: **ECS_E_ERRORFREE**
Remarks: There are no errors present.

2.6.2.2 Invalid Field Number

Message: **Invalid field number**
Constant: **ECS_E_FLDNUM**
Remarks: The value of specified field doesn't fall into valid range between 1 and **FieldCount**, which is obtained by calling **EcsGetFieldCount** subroutine, and contains maximum number of fields on current screen.

See: **EcsReadField, EcsGetFieldAttribute, EcsGetFieldLength, EcsGetFieldPosition, EcsWriteField**

2.6.2.3 Attempt to Write Protected Field

Message: **Attempt to write protected field**
Constant: **ECS_E_PROTFLD**
Remarks: Subroutine **EcsWriteField** was invoked with a protected field specified by **FieldNumber**.

See: **EcsWriteField**

2.6.2.4 Invalid Line Number

Message: **Invalid line number**

Constant: **ECS_E_LINENUM**

Remarks: The specified line number was not within a valid range. For 5250 sessions, you can specify line numbers between 1 and 24 inclusive. With occasional error conditions, user can specify line number 25, only for 5250 sessions, when OIA contains an error message. For 3270 sessions, valid range will vary depending on the MOD of the current active session. These MOD (screen size) changes occur dynamically and care must be taken invoking **EcsGetLastNonBlankLineNumber** and **EcsReadLine**.

See: **EcsReadLine, EcsGetLastNonBlankLineNumber**

2.6.2.5 Invalid Screen Area Limits

Message: **Invalid screen area limits**

Constant: **ECS_E_SCRNAREA**

Remarks: The values of **Top**, **Bottom**, **Left**, and **Right** that specify an area that resides within screen buffer are not valid.

See: **EcsReadScreen, EcsReadColoredScreen**

2.6.2.6 (Row, Column) is Not on Screen

Message: **(Row, Column) is not on screen**

Constant: **ECS_E_ROWCOL**

Remarks: The input variables **Row** and **Column** specified a character position that was not within current screen buffer.

See: **EcsGetFieldNumber**

2.6.2.7 Failed to Send Keystroke(s)

Message: **Failed to send keystroke(s)**

Constant: **ECS_E_SENDKEY**

Remarks: This error condition can occur when EASEL application fails to send information to the host, and this event was detected by ECS.

See: **EcsPressKey, EcsTypeString**

2.6.2.8 Screen Contains No Fields

Message: **Screen contains no fields**

Constant: **ECS_E_NOFLDS**

Remarks: This error condition can occur when there are no fields on current screen and a subroutine that references field information is being invoked.

See: **EcsReadField, EcsGetFieldAttribute, EcsGetFieldLength, EcsGetFieldNumber, EcsGetFieldPosition, EcsWriteField**

2.6.2.9 Empty or Invalid WatchCommandString

Message: **Empty or invalid WatchCommandString**

Constant: **ECS_E_WATCH**

Remarks: This error condition will occur whenever the user specifies an invalid watch string and invokes the ECS Watch facility.

See: **EcsWatch, EcsWatchAndWait**

2.6.2.10 Cannot Initialize

Message: **Cannot initialize**

Constant: **ECS_E_CANTINIT**

Remarks: Could not initialize. This error indicates that required hardware or software is not installed. For example, the IBM Enhanced 3270/5250 Emulation Adapter is installed, but the emulator software is not resident.

2.6.2.11 Screen ID File Not Found

Message: **Screen ID file not found**

Constant: **ECS_E_SCRIDFILE**

Remarks: **ScreenIDFile** that contains screen information cannot be located.

See: **EcsEmulateAndWatch**

2.6.2.12 Screen ID File Has Illegal Syntax

Message: **Screen ID file has illegal syntax**
Constant: **ECS_E_BADSCRID**
Remarks: **ScreenIDFile** that contains screen information is corrupted.
See: **EcsEmulateAndWatch**

2.6.2.13 Not Enough Memory for Screen IDs

Message: **Not enough memory for screen IDs**
Constant: **ECS_E_NOMEMORY**
Remarks: Not enough memory to process ScreenIDFile information.
See: **EcsEmulateAndWatch**

2.6.2.14 Not Connected to an Active Session

Message: **Not connected to an active session**
Constant: **ECS_E_NOTCONN**
Remarks: This error condition will occur if user tries to invoke any ECS subroutine, without being connected to active session.
See: Most of ECS subroutines

2.6.2.15 Invalid Session Name

Message: **Invalid session name**
Constant: **ECS_E_SESSNAME**
Remarks: This error condition will occur if user specified invalid session name. A valid name must be one of the 3270/5250 host session short names configured under CM.
See: **EcsConnect, EcsGetSessionInfo**

2.6.2.16 Session is Already in Use

Message: Session is already in use

Constant: **ECS_E_INUSE**

Remarks: This error condition will occur if user tries to connect to a session already connected to another process.

See: **EcsConnect**

2.6.2.17 Not Initialized - Command Was Ignored

Message: **Not initialized - command was ignored**

Constant: **ECS_E_NOTINIT**

Remarks: This error condition will occur if **EcsConnect** was not the first ECS subroutine called. All other calls will invoke this error event, until **EcsConnect** gets called.

See: All ECS subroutines

2.6.2.18 Key Unavailable for Current Session Type

Message: **Key unavailable for current session type**

Constant: **ECS_E_WRONGKEY**

Remarks: This error condition will occur if valid value is passed to input variable **KeyCode** of **EcsPressKey** subroutine, but it can only be applied to either 3270 or 5250 session, and the current session is of the inappropriate type.

Remarks: **EcsPressKey**

2.6.2.19 Not Connected to a 3270 Session

Message: **Not connected to a 3270 session**

Constant: **ECS_E_NOT3270SESSION**

Remarks: This error condition will occur if user invokes ECS 3270 session subroutines, while being connected to 5250 session.

See: **EcsEmulate, EcsEmulateAndWatch, EcsGetSessionStatus, EcsSetEmulatorWindow**

2.6.2.20 Multiple Watches Are Not Supported

Message: **Multiple watches are not supported**

Constant: **ECS_E_MULTWATCH**

Remarks: This error condition will occur if user invokes EcsWatch or EcsWatchAndWait subroutines, with one of them already in process. This can occur if EcsWatch or EcsWatchAndWait are invoked with AutoWaitFlag being set to ON, which will automatically call EcsWatchAndWait.

See: **EcsSetAutoWait, EcsWatch, EcsWatchAndWait**

2.6.2.21 Indicator is Unavailable for Current Session Type

Message: **Indicator is unavailable for current session type**

Constant: **ECS_E_BADINDICATOR**

Remarks: This error condition will occur if user specifies invalid value for input variable **Indicator** of **EcsCheckIndicator** subroutine, or a 3270 indicator is specified and the current session is 5250, or vice versa.

See: **EcsCheckIndicator**

2.6.2.22 Cannot Load EHLLAPI Module

Message: **Cannot load EHLLAPI module**

Constant: **ECS_E_CANTLOAD**

Remarks: This error condition will occur if EASEL OS/2 EE cannot locate ACS3EHAP.DLL file.

2.6.2.23 Cannot Create Thread for Watch

Message: **Cannot create thread for watch**

Constant: **ECS_E_NOTHREAD**

Remarks: Internal PM call failed. Call your service representative.

See: **EcsWatch, EcsEmulateAndWatch**

2.6.2.24 Invalid Field Attribute Specified

Message: **Invalid field attribute specified**

Constant: **ECS_E_FLDATTR**

Remarks: This error condition will occur if user specifies invalid value for input variable **FieldAttribute** of **EcsGetFieldAttribute** subroutine.

See: **EcsGetFieldAttribute**

2.6.2.25 Invalid Fill Switch Specified

Message: **Invalid fill switch specified**

Constant: **ECS_E_FILLSWITCH**

Remarks: This error condition will occur if user specifies invalid value for input variable **FillSwitch** of **EcsWriteField** subroutine, or if **ECS_FILLWITHNULLS** is specified for a 5250 session.

See: **EcsWriteField**

2.6.2.26 Key Code is Unrecognized

Message: **Key code is unrecognized**

Constant: **ECS_E_BADKEYCODE**

Remarks: This error condition will occur if user specifies invalid value for input variable **KeyCode** of **EcsPressKey** subroutine.

See: **EcsPressKey**

2.6.2.27 Cannot Determine Emulator's Status

Message: **Cannot determine emulator's status**

Constant: **ECS_E_NOSTATUS**

Remarks: Internal EHLLAPI call failed. Call your service representative.

See: **EcsEmulate**

2.6.2.28 Illegal Time ($0 < \text{time} < \text{Hour}$)

Message: **Illegal time ($0 < \text{time} < \text{hour}$)**

Constant: **ECS_E_BADTIME**

Remarks: This error condition will occur if user specifies invalid value for input variable **SettleTime** of **EcsWatchForIIOff** subroutine.

See: **EcsWatchForIIOff**

2.6.2.29 Empty WatchString

Message: **Empty WatchString**

Constant: **ECS_E_WATCHSTRING**

Remarks: This error condition will occur if user specifies invalid value for input variable **String** of **EcsWatchForStringAt** and **EcsWatchForStringNotAt** subroutines.

See: **EcsWatchForStringAt, EcsWatchForStringNotAt**

2.6.2.30 Bad Emulate Command

Message: **Bad emulate command**

Constant: **ECS_E_BADCOMMAND**

Remarks: This error condition will occur if user passes invalid input to the **EcsSetEmulatorWindow** subroutine.

See: **EcsSetEmulatorWindow**

2.6.2.31 Bad Emulate Command Combination

Message: **Bad emulate command combination**

Constant: **ECS_E_BADCOMBO**

Remarks: This error condition will occur if user passes invalid input to the **EcsSetEmulatorWindow** subroutine, consisting of incompatible window commands.

See: **EcsSetEmulatorWindow**

2.6.2.32 No Error for Specified Error Code

Message: **No error for specified error code**

Constant: **ECS_E_NOERROR**

Remarks: Will be expanded for additional error checking in the future.

2.7 Reserved Variables and Constants

2.7.1 Global Variables

The following global variables are used in ECS. Be sure your program does not contain any of these identifiers, except when referencing the ECS.

Global Variable Name	Type	Initial Value
EcsAutoWaitFlag	boolean	false
EcsWaitGaveUp	boolean	false
EcsTimeLimit	integer	3000 (3 sec.)
EcsWatchCommandString	string	"" (empty)

2.7.2 Reserved Constants

The following reserved constants are used in ECS. Be sure that your program does not contain any of these identifiers, except when referencing the ECS.

ECS_ATT	ECS_CMD0
ECS_AUTOWAITOFF	ECS_CMD1
ECS_AUTOWAITON	ECS_CMD2
ECS_BACK_SPACE	ECS_CMD3
ECS_BACK_TAB	ECS_CMD4
ECS_CLEAR	ECS_CURSR_SEL
ECS_CMD	ECS_DBL_DOWN_ARROW
ECS_CMD1	ECS_DBL_LEFT_ARROW
ECS_CMD2	ECS_DBL_RIGHT_ARROW
ECS_CMD3	ECS_DBL_UP_ARROW
ECS_CMD4	ECS_DELETE
ECS_CMD5	ECS_DEV_CNCL
ECS_CMD6	ECS_DOWN_ARROW
ECS_CMD7	ECS_DUP
ECS_CMD8	ECS_E_BADCOMMAND
ECS_CMD9	ECS_E_BADCOMBO
ECS_CMD10	ECS_E_BADINDICATOR
ECS_CMD11	ECS_E_BADKEYCODE
ECS_CMD12	ECS_E_BADSCRID
ECS_CMD13	ECS_E_BADTIME
ECS_CMD14	ECS_E_CANTINIT
ECS_CMD15	ECS_E_CANTLOAD
ECS_CMD16	ECS_E_ERRORFREE
ECS_CMD17	ECS_E_FILLSWITCH
ECS_CMD18	ECS_E_FLDATTR
ECS_CMD19	ECS_E_FLDNUM

ECS_E_INUSE	ECS_HEX
ECS_E_LINENUM	ECS_HOME
ECS_E_MULTWATCH	ECS_I_II
ECS_E_NOERROR	ECS_I_MW
ECS_E_NOFLDS	ECS_I_SA
ECS_E_NOMEMORY	ECS_I_X
ECS_E_NOSTATUS	ECS_IDENT
ECS_E_NOT3270SESSION	ECS_IIOFF
ECS_E_NOTAVAIL	ECS_IION
ECS_E_NOTCONN	ECS_INSERT
ECS_E_NOTHREAD	ECS_INSERT_TOGGLE
ECS_E_NOTINIT	ECS_INTENSIFIED
ECS_E_PROTFD	ECS_IS_INTENSIFIED
ECS_E_ROWCOL	ECS_IS_MODIFIED
ECS_E_SCRIDFILE	ECS_IS_NONDISPLAY
ECS_E_SCRNAREA	ECS_IS_PROTECTED
ECS_E_SENDKEY	ECS_LEFT_ARROW
ECS_E_SESSNAME	ECS_MODIFIED
ECS_E_WATCH	ECS_MWOFF
ECS_E_WATCHSTRING	ECS_MWON
ECS_E_WRONGKEY	ECS_NEW_LINE
ECS_EMULATEWATCHDONE	ECS_NOFILL
ECS_EMULATEWATCHERROR	ECS_NONDISPLAY
ECS_END	ECS_NOT_INTENSIFIED
ECS_ENTER	ECS_NOT_MODIFIED
ECS_ERASE_EOF	ECS_NOT_NONDISPLAY
ECS_ERASE_INPUT	ECS_NOT_PROTECTED
ECS_EW_ACTIVATE	ECS_PA1
ECS_EW_DEACTIVATE	ECS_PA2
ECS_EW_MAKE_INVISIBLE	ECS_PA3
ECS_EW_MAKE_VISIBLE	ECS_PF1
ECS_EW_MAXIMIZE	ECS_PF2
ECS_EW_MINIMIZE	ECS_PF3
ECS_EW_RESTORE	ECS_PF4
ECS_FIELD_EXIT	ECS_PF5
ECS_FIELD_MARK	ECS_PF6
ECS_FIELD_MINUS	ECS_PF7
ECS_FIELD_PLUS	ECS_PF8
ECS_FILLWITHBLANKS	ECS_PF9
ECS_FILLWITHNULLS	ECS_PF10
ECS_FT_ALPHA_ONLY	ECS_PF11
ECS_FT_ALPHANUMERIC	ECS_PF12
ECS_FT_DIGITS_ONLY	ECS_PF13
ECS_FT_MAG_STRIPE	ECS_PF14
ECS_FT_NUMERIC_ONLY	ECS_PF15
ECS_FT_NUMERIC_SHIFT	ECS_PF16
ECS_FT_RESERVED	ECS_PF17
ECS_FT_SIGNED_NUMERIC	ECS_PF18
ECS_HELP	ECS_PF19

ECS_PF20	ECS_TEST
ECS_PF21	ECS_UP_ARROW
ECS_PF22	ECS_WATCHDONE
ECS_PF23	ECS_WATCHERROR
ECS_PF24	ECS_WATCHGAVEUP
ECS_PRINT	ECS_XBADCRD
ECS_PROTECTED	ECS_XBADFUN
ECS_RECORD_BACKSPACE	ECS_XCLOCK
ECS_RESET	ECS_XCOMMCK
ECS_RIGHT_ARROW	ECS_XGOAWAY
ECS_ROLL_DOWN	ECS_XKEY
ECS_ROLL_UP	ECS_XMECHCK
ECS_SAOFF	ECS_XMORE
ECS_SAON	ECS_XNOFUN
ECS_SS_APPLICATION	ECS_XNOPRT
ECS_SS_CONTROL	ECS_XNOSYM
ECS_SS_DISCONNECTED	ECS_XNUM
ECS_SS_UNOWNED	ECS_XPROGCK
ECS_ST_3270_HOST	ECS_XPRTBSY
ECS_ST_3270_PRINTER	ECS_XPRTVRB
ECS_ST_5250_HOST	ECS_XREADY
ECS_ST_5250_PRINTER	ECS_XSYSTEM
ECS_ST_PC	ECS_XUNAUTH
ECS_ST_UNKNOWN	ECS_XUNKNOWN
ECS_SYS_REQ	ECS_XWHAT
ECS_TAB	ECS_XWHTNUM

2.8 Sample Program Using 3270/5250 Screens

Sample 3270 Screen "A":

MIN01001	ABC MANUFACTURING COMPANY
REPORT SELECTION	
<u>Report Name</u>	<u>ID Number</u>
Production Statistics	PS4471
Defect Reporting	DF1678
Warehouse Inventory	WI3672
Manufacturing Lead Time	MF2599
Enter Report ID Number: _____	

Sample 3270 Screen "B":

DF167801

Divn. : Acme ABC MANUFACTURING COMPANY
Report No: DF1678 DEFECTIVE PARTS REPORTING SYSTEM

Part Description	JAN	FEB	MAR	APR	MAY
3061 Cam Shaft	12	11	9	10	14
Bearing Ring	11	9	11	8	7
Time Gear - Large	11	19	14	16	15
Time Gear - Small	12	15	16	18	14
Cam Shaft Dowel Pin	15	16	13	16	15
Cam Shaft Lock Nut	9	11	12	12	13
Valve Bushing	12	9	7	9	19
TOTAL	82	90	82	89	97

- More -

Cmd1 = NEXT PAGE

Cmd7 = RETURN TO MENU

2.8.1 Sample Program Using 3270/5250 Support

```
screen size device units      # Use resolution of device
font "system"                # Use system font

#
# INCLUDE FILE REFERENCE
#

include "eslcmrv.inc"        # ECS declarations file

include "accel.inc"          # Include accelerator keys

include "message.inc"        # Include message box
                             # subroutines

#
# VARIABLE DEFINITION
#

boolean variable  More is false

string variable  ReplyIs      # contains reply
                  MessageIs    # contains message
                  SessionShortName # contains session name
                  String        # contains input string
                  Watch1        # contains watch string
                  TextRegion    # contains name of text
                             # region
                  LineContents  # contains line contents

integer variable  Top          # specifies top margin
                  Bottom        # specifies bottom margin
                  Left          # specifies left margin
                  Right         # specifies right margin
                  Row           # specifies row number
                  SettleTime     # specifies time in sec.
                  KeyCode        # contains function key
                  TimeLimit      # specifies time in sec.
```



```
# BAR DEFINITION
```

```
action bar ActBar is
```

```
    pulldown File text "~File"
```

```
        disabled choice Open text "~Open..."
```

```
        disabled choice New text "~New"
```

```
        separator
```

```
        disabled choice Save text "~Save"
```

```
        disabled choice SaveAs text "Save ~As..."
```

```
        separator
```

```
        choice Exit text "E~xit\tF3"
```

```
            accelerator is KEY_F3
```

```
    end pulldown
```

```
    pulldown ReportList text "~Reports"
```

```
        choice Prodstat text "~Production Stats"
```

```
            parameter is "PS4471"      # This is the ID No. (shown in
```

```
            # Sample Screen A)
```

```
        choice Defect text "~Defect Reports"
```

```
            parameter is "DF1678"      # This is the ID No. (shown in
```

```
            # Sample Screen A)
```

```
        choice Warehouse text "~Warehouse Inventory"
```

```
            parameter is "WI3672"      # This is the ID No. (shown in
```

```
            # Sample Screen A)
```

```
        choice Manufacturing text "~Manufacturing"
```

```
            parameter is "MF2599"      # This is the ID No. (shown in
```

```
            # Sample Screen A)
```

```
    end pulldown
```

```
end action bar
```

```
#
```

```
# OBJECT DEFINITIONS
```

```
#
```

```
white primary region Primary size 500 350 at position 75 75
```

```
    title bar "ABC Manufacturing Company"
```

```
    system menu
```

```
    action bar ActBar
```

```
    minimize button
```

```
    maximize button
```

```
#
```

```
# Establish a class of items that includes Report choices:
```

```
#
```

```
item class RptSelect is Prodstat Defect Warehouse Manufacturing
```

```

#
# Create a textual region to show the selected report:
#

invisible blue textual region Text
    window size 60 columns 17 lines
    at position 120 5 in desktop
    title bar "Requested Report"
    system menu
    vertical scroll bar
    border
    font "asc814"

#
# SUBROUTINES
#

# Copy current screen into Text textual region, and check if
# there are more screens to come:

subroutine ReadReportPage () is
    copy 1 to Top          # Initialized margins of screen
    copy 24 to Bottom      # buffer to be read into textual
    copy 1 to Left         # region Text
    copy 80 to Right
    copy "Text" to TextRegion # Initialized input variable
    call EcsReadScreen( Top,Left,Bottom,Right, # Reads
                        TextRegion )           # current
                                                # screen

    copy 22 to Row
    call EcsReadLine( Row, LineContents )
    switch LineContents is
        case char "- More -" is
            copy true to More
        default is
            copy false to More
    end switch

# At first page of report, read all pages into Text textual
# region:

subroutine GetPages () is
    call ReadReportPage()
    call EcsClearWatch( Watch1 ) # Clear watch string
    copy 4000 to SettleTime      # Reset watch string
    call EcsWatchForIIOff( SettleTime, Watch1 )

```

```

while (More) loop
    copy ECS_CMD1 to KeyCode    # Get next page if "- More -"
    call EcsPressKey(KeyCode)   # is displayed
    call EcsWatchAndWait( TimeLimit, Watch1 )
    if ( EcsWaitGaveUp ) then
        copy false to More
    else
        call ReadReportPage()
    end if
end loop

copy ECS_CMD7 to KeyCode          # Return to menu screen
call EcsPressKey( KeyCode )

#
# Log off from the AS/400:
#

    # Code to log off from AS/400 should be placed here.

:

#
# RESPONSE DEFINITIONS
#

response to start

    copy "A" to SessionShortName
    call EcsConnect( SessionShortName )

    # Code to log on to the AS/400 should be placed here.

:

response to item RptSelect          # Class of reports
    call EcsClearWatch( Watch1 )
    copy 7000 to TimeLimit
    copy 5000 to SettleTime
    call EcsWatchForIIOff( SettleTime, Watch1 )
    copy parameter to String
    call EcsTypeString( String )
    call EcsPressENTER()
    call EcsWatchAndWait( TimeLimit, Watch1 )

```

```

if ( EcsWaitGaveUp ) then                                # Put up message box
    copy "Could not get report " String "." to MessageIs
    copy (ReplyToMessage("System Problems", MessageIs,
                        MessageOK, MessageOK, MessageIconHand))
                        to ReplyIs
    make Text invisible
else
    call GetPages()
    make Text visible
end if

response to item Exit
    call Logoff()
    call EcsDisconnect()
    exit

response to Primary on close
    call Logoff()
    call EcsDisconnect()
    exit

```

Chapter 3. Permanent and Temporary Dialogs

3.1 Overview

You can now specify whether a dialog box or dialog region is permanent or temporary. Windows for temporary objects are created when the objects become visible, and deleted when the objects become invisible. You can reduce the number of windows by keeping dialog boxes and dialog regions temporary, and making them invisible when you no longer need to display or query them.

This change affects the EASEL OS/2 EE syntax and the Layout/CUA tool.

Note: This option is not valid for primary regions, which should always be visible.

3.2 EASEL OS/2 EE Syntax

The following changes have been made to EASEL OS/2 EE:

- The syntax of the **dialog box** definition has been revised. The syntax of the **dialog region** definition has been revised.
- Two new action statements, **make permanent** and **make temporary**, have been added.
- A new runtime command line option, **-p**, has been added.

3.2.1 dialog box Definition

The following model replaces the **dialog box** Definition model. The following text is an addition, not a replacement:

Model

```
[ enabled | disabled ] [ visible | invisible ]  
[ temporary | permanent ] [ modal | modeless ] dialog box DB_NAME  
size WIDTH HEIGHT  
at [position] X Y  
[parameter is PAR_STRING]  
[ [dialog] border]  
[title bar TITLE_STRING]  
[system menu]
```

<i>Use of Keywords:</i>	temporary specifies that the dialog box will be temporary. The window will be created when the dialog box is made visible. The window will be deleted when the dialog box is made invisible. This keyword enables you to avoid reaching the OS/2 window limit in an application with many dialog boxes or dialog regions. permanent specifies that the dialog box will be permanent. Data about the dialog box will be preserved while the box is invisible.
-----------------------------	--

Remarks: The default permanency is **temporary**. The runtime option **-p** (permanent) overrides the **temporary** keyword for all dialog boxes and dialog regions.

Dialog controls inherit the permanency of their parent. You can write responses to objects and their children even if they are temporary and invisible, but these responses will not be taken until the objects are made visible.

When a temporary and invisible dialog box is made visible again, you cannot rely on the values of the attributes of the dialog box or its controls. If you want to redisplay a temporary and invisible dialog box, reset the attribute values after the object is made visible or permanent. To avoid momentary display of incorrect values, you can make the object permanent, then set the attributes, and then make the object visible.

Note: While a dialog box is temporary and invisible, queries and action statements on the dialog box and its children will produce unpredictable results. If you want to use inquiry functions or action statements on an invisible object, use the **make permanent** action statement to make the object permanent before making it invisible.

The **temporary** keyword reduces the number of windows only if objects are made invisible.

Errors: If an inquiry function or action statement references a temporary and invisible dialog box, the following error might result:

ERROR : window handle of object DB_NAME not found.

where ERROR is the specific error that occurred.

See Also: **make permanent** and **make temporary** Action Statements Runtime Option

3.2.2 dialog region Definition

The following model replaces the dialog region Definition model. The following text is an addition, not a replacement:

Model

```
[ enabled | disabled ] [ visible | invisible ]
[primary] [ temporary | permanent ] dialog region DR_NAME
size XSIZE YSIZE
at { [position] X Y | default position }
[in PARENT_NAME1 [in PARENT_NAME2]... ]
[priority is PRIORITY_NO]
[parameter is PAR_STRING]
[ [size] border]
[title bar TITLE_STRING]
[system menu]
[horizontal scroll bar [scroll by SCROLL_INC] ]
[vertical scroll bar [scroll by SCROLL_INC] ]
[action bar AB_NAME]
[minimize button [using MINIMIZE_ICON_FILE] ]
[maximize button]
```

*Use of
Keywords:*

temporary specifies that this region is temporary. The window will be created when the dialog region is made visible. The window will be deleted when the dialog region is made invisible. This keyword enables you to avoid reaching the OS/2 window limit in an application with many dialog boxes or dialog regions.

permanent specifies that the dialog region will be permanent. Data about the dialog region will be preserved while the region is invisible.

Remarks:

The default permanency is **temporary**. The runtime option **-p** (permanent) overrides the **temporary** keyword for all dialog boxes and dialog regions.

Dialog controls inherit the permanency of their parent. You can write responses to objects and their children even if they are temporary.

When a temporary and invisible dialog region is made visible again, you cannot rely on the values of the attributes of the dialog region or its controls. If you want to redisplay a temporary and invisible dialog region, reset the attribute values after the object is made visible or permanent. To avoid momentary display of incorrect values, you can make the object permanent, then set the attributes, and then make the object visible.

Note: While a dialog region is temporary and invisible, queries and action statements on the dialog region and its children will produce unpredictable results. If you want to use inquiry functions or action statements on an invisible object, use the **make permanent** action statement to make the object permanent before making it invisible.

The **temporary** keyword reduces the number of windows only if objects are made invisible.

Errors: If an inquiry function or action statement references a temporary and invisible dialog region, the following error might result:

ERROR : window handle of object DR_NAME not found.
where ERROR is the specific error that occurred.

See Also: **make permanent** and **make temporary** Action Statements Runtime Option

3.2.3 make permanent and make temporary Action Statements

The following action statements are new:

Purpose: To change the permanency of a dialog box or dialog region.

Model

make { DB_NAME | DR_NAME } { permanent | temporary }

Where: DB_NAME is the identifier for a dialog box.
DR_NAME is the identifier for a dialog region.

Remarks: When an invisible object is made temporary, its window is deleted.

If you want to use inquiry functions on an invisible dialog box or dialog region, make the object permanent before making it invisible. To reduce the number of windows, make an object temporary and invisible once its information is no longer needed.

When a temporary and invisible object is made visible again, you cannot rely on the values of the attributes of the object or its dialog controls. If you want to redisplay a temporary and invisible object, reset the attribute values after the object is made visible or permanent. To avoid momentary display of incorrect values, you can make the object permanent, then set the attributes, and then make the object visible.

Examples: response to Cancel
 make DialogBox1 permanent

response to UpdateComplete
 make DialogBox1 temporary

Errors: If the **make permanent** or **make temporary** action statement references an object that does not exist, the following error results:

error changing object permanency: object OBJ_NAME not found.

See Also: **dialog box** and **dialog region** Definitions

3.3 Runtime Option

A new command line option can be issued on the command line at runtime or included on the eslrn line of the EASEL.CMD file. This option enables existing EASEL OS/2 EE applications to function without modification:

-p specifies that all dialog boxes and dialog regions are permanent, even if they were defined as temporary. *Initial value:* Not specified.

Note: Because dialog boxes and dialog regions are temporary by default, this option is required for backward compatibility.

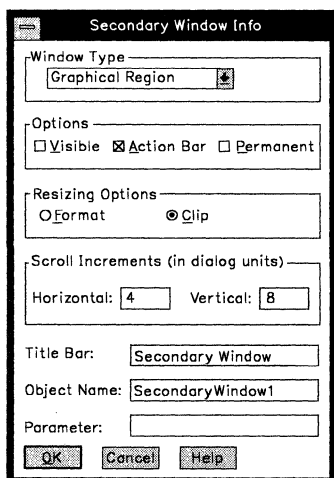
The following example shows how to run existing applications, where FILENAME is replaced with the name of the particular EASEL OS/2 EE .EBI filename.

```
ease1 FILENAME -p
```

3.4 Layout/CUA

3.4.1 Secondary Window

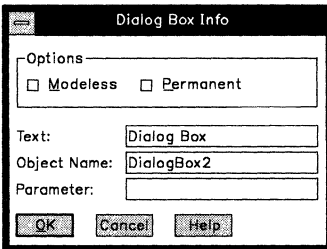
The following change has been made to the Secondary Window Info dialog box:



The Permanent check box allows you to specify whether or not a secondary dialog region is permanent. Windows for temporary (non-permanent) regions are created when the regions become visible, and deleted when the regions become invisible. The permanent check box is valid only for Dialog Region window types.

3.4.2 Dialog Box

The following change has been made to the Dialog Info dialog box:



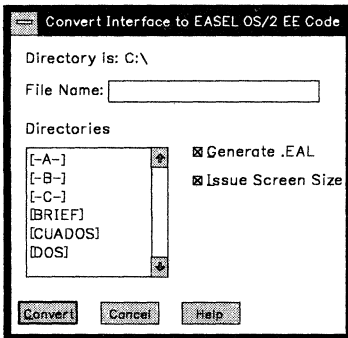
The Permanent check box allows you to specify whether or not the dialog box is permanent. Windows for temporary (non-permanent) dialog boxes are created when the dialog boxes become visible, and deleted when the dialog boxes become invisible.

Chapter 4. Miscellaneous Enhancements

4.1 Scaling in Layout/CUA

Before you convert a Layout/CUA interface, you can now specify whether or not windows should be scaled for different runtime graphics resolutions.

The following change has been made to the Convert Interface to EASEL OS/2 EE Code dialog box.



Use the Issue Screen Size check box to indicate whether or not you want a screen size environment declaration included in the .EAL file. If the box is checked, EASEL OS/2 EE will scale all windows, except those specified with the Clip resizing option or the EASEL OS/2 EE no scale keyword, for different runtime (end-user) graphics resolutions.

For example, Issue Screen Size generates the following EASEL OS/2 EE code for a VGA development system:

```
screen size 640 480
```

If the end-user's system has a different resolution, the windows will occupy the same proportion of the screen as on the development system.

Note: If you select the Issue Screen Size check box, windows specified with the Clip window resizing option will be scaled, but their contents will not. Use the Format resizing option for all windows if you select the Issue Screen Size check box.

4.2 New File I/O Error Messages

4.2.1 Open Subroutine Error Messages

The following new messages can be returned by the **OpenFile()** subroutine.

errorlevel	Error Message/Explanation
30	Attempt to open a file in a directory that doesn't exist. This error was covered previously by errorlevel 4 (Attempt to open a file that doesn't exist).
33	Attempt to open a file that has been opened by another process. Retry the command later.
34	Attempt to open read-protected file with mode "r". Check the access rights of the file's directory.

4.2.2 Input Subroutines Error Messages

The following new messages can be returned by any of the input subroutines, **ReadLineNumber()**, **ReadNext()**, **ReadNextRecord()**, and **ReadRecordAtLine()**.

errorlevel	Error Message/Explanation
19	(New text.) The section of the file you are trying to access is locked by another process. This file has been locked by some other application.
31	Read access denied. Should never happen during input.

4.2.3 Output Subroutines Error Messages

The following new messages can be returned by either of the output subroutines, **WriteString()** and **WriteRecord()**.

errorlevel	Error Message/Explanation
16	(New text.) Unknown write error. An unknown error occurred while writing to a file.
19	The section of the file you are trying to access has been locked by another process. This file has been locked by some other application.
32	Write access denied. Should never happen during output.

Index

A

- action statements
 - make permanent 3-7
 - make temporary 3-7

B

- built-in functions 2-29

C

- communicating with the host 2-87
- components 2-10
 - ECS include file 2-10
 - ECSFIELD.EXE 2-11
- connecting sessions 2-34
- connecting to host sessions 2-15
 - EcsConnect 2-16
- controlling the emulator
 - window 2-100

D

- datatype is attribute 1-16
- DBCS support
 - eng option 1-24
 - sosi option 1-34, 1-40
 - APPC subroutines 1-32
 - APPC support 1-32
 - audio capture and playback 1-31
 - bisected characters 1-27, 1-34, 1-38, 1-40, 1-44
 - block operations 1-18
 - built-in functions 1-20, 1-21, 1-22
 - business graphics 1-29
 - characters 1-7
 - code page support 1-6
 - column position 1-17
 - dbcs field type 1-26
 - default font 1-12

DBCS support (*continued*)

- device support limitations 1-4
- EmulateAndWatch 1-39
- entry field data types 1-16, 1-25
- escape sequences 1-8, 1-20
- EslSetEgaColor function 1-26
- extract from 1-10
- FIELDS52.EXE 1-40
- FIELDS.EXE 1-34
- file i/o 1-26
- FindLastNonBlankLine 1-39
- fonts 1-12
- GetFieldAttributes 1-38
- Init3270 1-37
- input function 1-21
- installation 1-3
- keyboard status display
 - area 1-15, 1-24
- language elements 1-6
- Layout/CUA 1-24
- length of function 1-21
- library functions 1-26
- line continuation 1-7
- machine requirements 1-3
- match clauses 1-23
- mnemonics 1-8, 1-24
- M52_Connect 1-43
- M52_FindLastNonBlankLine 1-45
- M52_GetFieldAttributes 1-44
- M52_Init 1-43
- M52_PressKey 1-45
- M52_PrintScreen 1-45
- M52_WatchForChar 1-44
- M52_WatchForNotChar 1-44
- M52_WriteField 1-45
- null command 1-30
- plot statement 1-47
- printing and plotting 1-47
- Print3270Screen 1-39
- program requirements 1-3
- record definition 1-26
- return codes 1-32

DBCS support (*continued*)

- segment operations 1-19
- set command 1-30
- skip clause 1-10
- SO/SI printing 1-34, 1-40
- SQL support 1-30
- string operations 1-19
- strings 1-7, 1-8
- take clause 1-10
- text cursor 1-17
- textual regions 1-17
- touch screen 1-28
- type at function 1-17, 1-20
- WatchForChar 1-38
- WatchForNotChar 1-38
- WeekDay function 1-26
- WriteField 1-39
- xsize of function 1-22
- 3270 support 1-34
- 3270/5250 Support 1-46
- 5250 support 1-40

definition

- dialog box 3-3
- dialog region 3-5
- dialog box definition 3-3
- dialog region definition 3-5
- dialogs 3-1
- disconnecting from host
 - sessions 2-15
 - EcsDisconnect 2-16
- disconnecting sessions 2-34

E

EASEL OS/2 EE Communications

- Server (ECS) 2-10
 - using the ECS 2-11
- ECS include file 2-10
- ECS stimulus library
 - declaring 2-11
 - sample program 2-12
- ECS subroutines 2-28
 - built-in functions 2-29
 - communicating with host 2-87
 - connecting sessions 2-34
 - controlling the emulator
 - window 2-100

ECS subroutines (*continued*)

- disconnecting sessions 2-34
- examining the buffer 2-40
- miscellaneous
 - subroutines 2-109
- summary of 2-29
- synchronizing with host 2-64
- ECS watch facility 2-64
- EcsAutoWaitFlag variable 2-92
- EcsCheckIndicator
 - subroutine 2-109
- EcsClearWatch 2-29
- EcsClearWatch subroutine 2-66
- EcsConnect 2-16
- EcsConnect subroutine 2-34
- EcsDisconnect 2-16
- EcsDisconnect subroutine 2-37
- EcsEmulate subroutine 2-102
- EcsEmulateAndWatch
 - subroutine 2-104
- EcsEnterString subroutine 2-96
- EcsGetCurrentSession
 - subroutine 2-117
- EcsGetCursorPosition
 - subroutine 2-50
- EcsGetError subroutine 2-115, 2-119
- EcsGetFieldAttribute
 - subroutine 2-48
- EcsGetFieldCount subroutine 2-42
- EcsGetFieldLength subroutine 2-44
- EcsGetFieldNumber
 - subroutine 2-52
- EcsGetFieldPosition
 - subroutine 2-54
- EcsGetFieldType subroutine 2-46
- EcsGetLastNonBlankLineNumber 2-16
- EcsGetLastNonBlankLineNumber
 - subroutine 2-57
- EcsGetSessionInfo
 - subroutine 2-113
- EcsGetSessionStatus
 - subroutine 2-111
- EcsPressENTER subroutine 2-92
- EcsPressKey subroutine 2-89

- EcsReadColoredScreen
 - subroutine 2-61
- EcsReadField subroutine 2-40
- EcsReadLine 2-16
- EcsReadLine subroutine 2-55
- EcsReadScreen subroutine 2-59
- EcsSetAutoWait subroutine 2-87
- EcsSetEmulatorWindow
 - subroutine 2-100
- EcsTypeString subroutine 2-94
- EcsWatch subroutine 2-82
- EcsWatchAndWait 2-85
- EcsWatchAndWait subroutine 2-85
- EcsWatchForCursorAt 2-29
- EcsWatchForCursorAt
 - subroutine 2-70
- EcsWatchForCursorNotAt 2-29
- EcsWatchForCursorNotAt
 - subroutine 2-72
- EcsWatchForIIOff 2-29
- EcsWatchForIIOff subroutine 2-74
- EcsWatchForNotStringAt 2-29
- EcsWatchForScreenSize
 - subroutine 2-67
- EcsWatchForStringAt 2-29
- EcsWatchForStringAt
 - subroutine 2-76
- EcsWatchForStringNotAt
 - subroutine 2-79
- EcsWriteField subroutine 2-98
- ECS_E_BADCOMBO 2-127
- ECS_E_BADCOMMAND 2-127
- ECS_E_BADINDICATOR 2-125
- ECS_E_BADKEYCODE 2-126
- ECS_E_BADSCRID 2-123
- ECS_E_BADTIME 2-127
- ECS_E_CANTINIT 2-122
- ECS_E_CANTLOAD 2-125
- ECS_E_ERRORFREE 2-120
- ECS_E_FILLSWITCH 2-126
- ECS_E_FLDATTR 2-126
- ECS_E_FLDNUM 2-120
- ECS_E_INUSE 2-124
- ECS_E_LINENUM 2-121
- ECS_E_MULTWATCH 2-125
- ECS_E_NOERROR 2-128
- ECS_E_NOFLDS 2-122
- ECS_E_NOMEMORY 2-123
- ECS_E_NOSTATUS 2-126
- ECS_E_NOTCONN 2-123
- ECS_E_NOTHREAD 2-125
- ECS_E_NOTINIT 2-124
- ECS_E_NOT3270SESSION 2-124
- ECS_E_PROTFLD 2-120
- ECS_E_ROWCOL 2-121
- ECS_E_SCRIDFILE 2-122
- ECS_E_SCRNAREA 2-121
- ECS_E_SENDKEY 2-121
- ECS_E_SESSNAME 2-123
- ECS_E_WATCH 2-122
- ECS_E_WATCHSTRING 2-127
- ECS_E_WRONGKEY 2-124
- error handling 2-119
 - messages 2-120
 - status retrieval 2-119
- error messages 2-120
 - file i/o 4-3
- error status retrieval 2-119
- ESCFIELD.EXE
 - development tool 2-11
 - reading the report 2-19
 - retrieving field information with 2-19
 - sample output 2-21
 - using 2-19
- examining the buffer 2-40

F

- field types 2-17
- field-oriented screens 2-16
 - attributes 2-17
 - colors 2-17
 - field types 2-17
 - Operator Information Area 2-18
 - 3270 screen buffer 2-16
 - 5250 screen buffer 2-16
- file i/o error messages 4-3
- functions 2-29

G

global variables 2-129

I

installation
 DBCS support 1-3

K

keyboard status display area 1-15

L

Layout/CUA 3-9
 permanent checkbox 3-9, 3-10
 scaling 4-2
 screen size 4-2
line-oriented screens 2-16
 accessing 2-16

M

make permanent 3-7
make temporary 3-7
miscellaneous subroutines 2-109

O

object permanency 3-7
OpenFile() 4-3
operator information area 2-18

P

-p runtime option 3-8
permanency 3-7
permanent dialogs 3-1
permanent runtime option 3-8

R

ReadLineNumber() 4-3
ReadNextRecord() 4-3
ReadNext() 4-3

ReadRecordAtLine() 4-3
reserved constants 2-129
reserved variables 2-129
 global 2-129
response to stimulus
 statements 2-82
retrieving field information 2-19
runtime option 3-8

S

sample program 2-12, 2-132
scaling in Layout/CUA 4-2
screen concepts 2-15
screen size 4-2
session concepts 2-15
session names 2-15
session states 2-15
 connected, active 2-15
 connected, not active 2-15
 disconnected 2-15
stimulus statements
 response to 2-82
subroutines
 EcsCheckIndicator 2-109
 EcsClearWatch 2-66
 EcsConnect 2-34
 EcsDisconnect 2-37
 EcsEmulate 2-102
 EcsEmulateAndWatch 2-104
 EcsEnterString 2-96
 EcsGetCurrentSession 2-117
 EcsGetCursorPosition 2-50
 EcsGetError 2-115, 2-119
 EcsGetFieldAttribute 2-48
 EcsGetFieldCount 2-42
 EcsGetFieldLength 2-44
 EcsGetFieldNumber 2-52
 EcsGetFieldPosition 2-54
 EcsGetFieldType 2-46
 EcsGetLastNonBlankLineNumber 2-
 EcsGetSessionInfo 2-113
 EcsGetSessionStatus 2-111
 EcsPressENTER 2-92
 EcsPressKey 2-89
 EcsReadColoredScreen 2-61

subroutines (*continued*)

- EcsReadField 2-40
- EcsReadLine 2-55
- EcsReadScreen 2-59
- EcsSetAutoWait 2-87
- EcsSetEmulatorWindow 2-100
- EcsTypeString 2-94
- EcsWatch 2-82
- EcsWatchForCursorAt 2-70
- EcsWatchForCursorNotAt 2-72
- EcsWatchForIIOff 2-74
- EcsWatchForScreenSize 2-67
- EcsWatchForStringAt 2-76
- EcsWatchForStringNotAt 2-79
- EcsWriteField 2-98

synchronizing with the host 2-64

T

temporary dialogs 3-1

U

using the ECS 2-11

W

window handle 3-4, 3-6

WriteRecord() 4-4

WriteString() 4-4

Numerics

3270 screen buffer 2-16

5250 screen buffer 2-16



SC38-7511-00



Printed in U.S.A.